ONTOLOGY DESIGN THROUGH MODULAR REPOSITORIES

Ali Hashemi and Michael Gruninger
Semantic Technologies Laboratory, University of Toronto, 5 King's College Circle, Toronto, Canada

Keywords: Ontologies, Ontology design, Knowledge representation, Knowledge engineering.

Abstract: Many real world problems require a language at least as expressive as first order logic, yet there exist many barriers to the generation of first-order ontologies. One of the biggest hurdles is the specification of axioms that capture the intended semantics of a user's concepts. This paper presents an ontology design algorithm enabled by modular ontology repositories that consist of theories organized into disjoint hierarchies, each of which is a set of nonconservative extensions. The ontology design algorithm provides axiomatizations of relations by eliciting intended models from the users, identifying the strongest theories in the repository that are satisfied by the intended models, and incorporating user feedback to verify the proposed set of axioms. This approach emphasizes the communication of semantics rather than syntax via concrete examples, allowing users to express intuitions about their domains without extensive background in the intricacies of formal languages.

1 INTRODUCTION & MOTIVATION

Many of the problems encountered in realistic applications require ontologies that are specified in a language with at least first-order expressivity. Nevertheless, first-order logic poses significant hurdles for many subject matter experts. Few have the adequate training or familiarity in first-order logic to express their ideas with facility in the language. The syntax and grammar of most implementations of first-order logic may often appear unintuitive and unwieldy. Compounding these barriers is also a paucity of guidelines and tools to support the generation of axioms; no best practices exist to help designers formulate axioms. Moreover, once expressed, it may be difficult to gauge the quality of the axioms.

Indeed, as Hou et al have noted, "research in classifying and representing axioms in a user-friendly way has been relatively sparse in the knowledge-base system community" (Hou et al., 2005). Several notable attempts have been made in this regard, both attempting to develop patterns and templates for axiom formulation based on existing ontologies (Hou et al., 2005; Staab and Maedche, 2000). More recent work has focused on identifying "design patterns" which ontology designers may reuse (Presutti and Gangemi, 2008). Yet these works focus on directly representing the axioms to user, for example by translating logical axioms into English - i.e. www.ontologydesignpatterns.org. This work differs drastically as it focuses on communicating with the ontology designer at the semantic level. In many fields a domain expert might be more comfortable specifying a relation via concrete positive and negative examples.

A versatile ontology design algorithm has been developed which allows ontology designers to circumvent the problem of becoming intimately familiar with a first order logic language, instead allowing them to focus on the semantics of what they wish to represent. We do so by requiring only two things: (i) that a domain expert be able to "draw" at least one representation of a model for relation to be defined; (ii) the subject matter expert must be able to recognize whether models of existing ontologies are acceptable manifestations of their relation. This algorithm is enabled by incorporating two simple principles in the design of an ontology repository, allowing the repository to function as a "map" of known theories. The algorithm then traverses the repository to deliver the most appropriate axioms to the ontology designers.

The algorithm rests on the relationship between a theory and its model (in the Tarskian sense). The word *model* as used throughout this paper, refers to the set of objects in a domain of discourse which satisfy the axioms. While in general, the correspondence between a set of models and a set of axioms that are

Hashemi A. and Gruninger M. (2009).
ONTOLOGY DESIGN THROUGH MODULAR REPOSITORIES.
In *Proceedings of the International Conference on Knowledge Engineering and Ontology Development*, pages 192-199
DOI: 10.5220/0002299201920199
Copyright © SciTePress

satisfied by them are many to many, we address this problem by exploiting properties of the accompanying repository. This paper will begin by briefly describing the essential components driving the algorithm, followed by the strategy employed to capitalize the relation between syntax and semantics. Next, the basic principles of the supporting ontology repository will be provided, followed by the ontology design algorithm and a sketch of its correctness proof. A brief use case will provide a concrete example of the algorithm in action.

2 MODELS AND LOGICAL STRUCTURE

We consider a formal ontology as in the sense provided in (Guarino, 1998). Namely, an ontology O_k, is a set of logical formulae in some language L that aim to capture the intended models of a particular conceptualization C. A model for the ontology is constructed via an interpretation I, assigning each element of the vocabulary V to the extensional structure of the conceptualization $S = \{D, R\}$ and thus to either elements of the domain D or the conceptual relations R (Guarino, 1998).

The approach taken here is to bypass the syntax and grammar of any particular formal language, and instead let users specify concepts extensionally. Particularly, if we focus our attention on the models of any theory, we notice that they contain particular structures. Take for example a Hasse diagram, which is often used to represent models for partially ordered sets. In these diagrams, nodes are taken to be elements from the domain of discourse, D and edges the less-than-or-equal-to *(leq)* relation (from R). These models implicitly capture the axioms which generated them; specifically, the way in which the edges and nodes are constructed reflect the associated poset axioms. While the mapping of nodes to say numbers, and edges to *leq* is achieved via a particular interpretation, the resultant model structures exist independently of the interpretation which assigned them to that conceptualization. Let us call this abstracted diagram a particular model structure (in this paper, we use the term *logical structure* interchangeably) for a theory. If we change our domain of discourse, the axioms for posets could equally well apply to a particular notion of time, or some versions of mereology.

We exploit exactly this notion of model structure to bypass the syntax of a formal language such as first-order logic and allow ontology designers to specify a particular relation via communication at the model representation level alone. Of course, there exists in general a many-to-many mapping between models and theories. Figuring out how to properly make the transition from a set of models to the appropriate theory is more involved. Once we start collecting and growing the acceptable models, the number of candidate theories is reduced. Moreover, by keeping track of those models which we *do not* want, we identify the unintended models. By collecting a coherent set of models that we want, and a set of those we don't want, in the limit we might reach a one-to-one mapping between a theory and a set of models.

The above is the key to our algorithm - we try to find the best match between a theory in a given repository and the set of (in)admissable models as identified by an ontology designer. Since the set of intended models, IM is not always immediately available, we elicit a subset from the user, UM. As noted above, it is not enough to jump from UM to a theory, T. Further interaction is required in which the system provides models SM of existing ontologies and the user identifies as either admissible or not. In this way we may be more confident in going from a set of models to a theory. There are a number of further nuances to this approach which will be elaborated in the rest of the paper.

3 REPOSITORY

Underlying the design algorithm of course is a modular ontology repository - this where candidate theories are selected from (Luettich and Mossakowski, 2004). Any repository satisfying the two basic principles discussed later in this section, allows a successful implementation of the algorithm (guarantees its correctness proof). For ease of understanding and concreteness, in this paper we present a particular repository design using the Common Logic Interchange Format (CLIF) (Delugach, 2007).

Any such repository should extend along two dimensions: one which we call *Abstraction Layers* and the other *Core Hierarchies.* Abstraction layers serve to significantly reduce conceptual clutter and help better delineate the types of theories being discussed; moreover they are essential in helping satisfy the second repository design criterion. Core Hierarchies gather theories as a map which the algorithm traverses.

3.1 Abstraction Layers

The repository serves as a sort of inverted upper ontology, since users are plugging in not necessarily to reuse *concepts* such as time and space, but to reuse the model structures of their underlying logical theories. In our example repository, we decided that mathematical theories corresponded to the lowest level of abstraction.

Theories formalizing the notions of orderings, groups, fields, geometries characterize a large family of logical and model structures that recur frequently in many applications. These theories form our base abstraction layer. Our next Abstraction Layer consists of theories from the traditional domain of upper ontologies - formalizing notions such as Space, Time, Mereotopology etc. Note that many of these concepts actually reuse logical structures from the base mathematical layer. Moving upwards thusly, we add layers which characterize more and more specialized concepts (say agents, or processes etc.)

The layers are connected to one another either via representation theorems or mapping axioms. Discussion of these links are outside the scope of this paper. For the purposes of the algorithm, it suffices to say that a repository ought be organized via some sort of abstraction layers, as each abstraction layer consists of Core Hierarchies, which drive the design process.

3.2 Core Hierarchies

As noted above, each layer is populated by a number of Core Hierarchies. A Core Hierarchy is a collection of modules (theories or set of axioms), which non-conservatively extend a conceptual domain. For example, the notion of a partial order may be non-conservatively extended in numerous ways from poset (one module) to lattice (another module) to Boolean lattice (yet another module). Of course, a repository may consist of more than one core hierarchy - thus our Abstraction Layer for mathematical theories consist of separate hierarchies for posets, geometries, groups, symmetries, fields etc.

The stipulation here that any compliant repository must satisfy is that no Core Hierarchies at the same level of abstraction may share a non-conservative extension with one another. If there exists a module within an abstraction layer which is a non-conservative extension of two hierarchies, then it suggests that the two hierarchies should in fact be combined as one.

In comparing two theories or modules in a core hierarchy from the repository, we may also say that one is *stronger* than another as follows. Given module A and B, if A is a non-conservative extension of B, then A is *stronger* than B by virtue of the fact that more theorems may be proven.

To recap, there were two requirements for any repository (modularity is taken as a given). First, modules should be linked in such a way as to form a *Core Hierarchy*, with the caveat of no shared non-conservative extensions at the same abstraction level. Following from this restriction, is that the repository should incorporate the notion of abstraction layer (the specific ordering of layers is left to the repository designer).

4 ALGORITHM

Now that the basic reference point from which theories will be selected has been exposited, we shall describe how such a repository may be leveraged. The algorithm consists of two parts, (i) elicitation of user models and (ii) the proposal of models for existing ontologies (see Figure 1.

The first component locates the user somewhere in the repository, providing “bounds” for theories which characterize the user’s intended models. Models derived from existing ontologies in the repository, coupled with user responses, tighten this bound, eventually selecting the strongest (if any) theories from the repository which capture the user’s intuition. The following sections will elaborate each component.

4.1 Elicitation of User Models

Acquiring user models necessitates that there exist a suitable representation for models of the concept or relation under consideration. Such a representation is not always obvious or available. However, any representation of a model is suitable so long as we are able to *unambiguously and repeatedly generate a complete diagram (the set of all positive and negative literals) from the depiction*. Consider again a Hasse diagram - there are certain conventions for interpreting the diagram - namely that edges are transitive and ordered spatially. So long as these conventions are explicitly communicated and understood, we may always generate the same complete diagram from a particular Hasse diagram, as in Figure 5.

The elicitation of models need not be restricted to Hasse diagrams, indeed changing the conventions of graph depictions might allow one to specify a model for a non-transitive relation. Alternatively, an auditory or tactile model depiction may be more natural for certain domains. For our proof of concept, we developed a simple piece of software allowing a user to specify binary relations using graphs. Users could also explicitly modify conventions for converting graphs into complete diagrams. Once a complete diagram for a model is generated, we identify which theories the model satisfies. In this way, we locate the

Inputs:

- At least one user generated model that is interpretable by the ontology design tool.
- Yes or no answers by the user to the models selected by the design tool

Outputs: The set of strongest axioms with respect to the repository that is consistent with the user's understanding as determined by the sets of accepted and rejected models.

1. Elicit-Models
2. Select-Models

Figure 1: The Algorithm *Axiom-Generation*.

initial candidate theories which may characterize the user's intended models.

4.2 Initializing in the Repository

To begin the algorithm, we require at least one (or more) user model, which is independently tested against theories in a single core hierarchy via a breadth-first specialization search[1]. We begin with the most general module, see if it is satisfied by the model, then move on to its children. If a particular theory is not satisfied, then it and all its children are pruned from the search tree. Moreover, since the hierarchies share no non-conservative extensions, we may explore them independently. We make use of an automated theorem prover or a satisfiability checker to verify that the provided models satisfy the theories.

We may then compare the theories which were satisfied by each user model. If no core-hierarchy had at least one theory which was satisfied by all user models, then the algorithm terminates - either the user is trying to formalize inconsistent models, or the repository has insufficient breadth. On the other hand, if there is at least one module in *some* core hierarchy which is satisfied by all models, then we may proceed to the second phase of the algorithm. We present here the algorithm in parts; while the steps are distributed through the text, they comprise a single algorithm.

The steps in Figure 2 first collect all the modules in hierarchy j satisfied by user model i in $Consistent\_UM_{i,j}$. We take its intersection and rid all the parents to yield the strongest theories in hierarchy j that are consistent with all the user models in $Consistent_j$. This set serves as the lower bound of candidate theories, the upper bound is simply the root of the hierarchy. We have thus located the user in the repository and passed the initiative to the software.

[1]We will use the following terminology: $CH_{m,j}$ denotes module m in hierarchy j, UM_i denotes user model i, $Above_K$ is the set of all modules connected to and directly above theory K, and $Below_K$ is the set of all modules connected to and directly below theory K.

1. Let $UM = UM_i$ the set of all models generated by user
2. **Breadth-First Specialization Search:** If $CH_{m,j}$ is satisfied by UM_i, then add $CH_{m,j}$ to $Consistent\_UM_{i,j}$.
3. **Termination Condition:**
   (a) If for any UM_i every $Consistent\_UM_i$ is empty, then **End Algorithm.**
   (b) If for any j, the intersection of all $Consistent\_UM_{i,j}$ is empty, then **End Algorithm**.
4. **Initialize:** Let $Consistent_j = \bigcap_j Consistent\_UM_{i,j}$
5. **Rid Parents:** For every element $CH_{m,j} \in Consistent_j$ if its child is in $Consistent_j$ then remove $CH_{m,j}$ from $Consistent_j$

Figure 2: The Algorithm *Elicit-Models*.

One problem with trying to derive an underlying theory by examining extensional models is that the models might exhibit accidental properties. For example, if asked to draw a *triangle*, many will inadvertently draw an equilateral or isosceles triangle. To account for this condition, we must explore more general theories and see if a model of them is acceptable to the user. If so, then the weaker theory is appropriate.

4.3 Select Models

Once user models have been mapped to the hierarchy, there are two scenarios: either there is a unique lower bound or there are multiple candidate theories. If unique, the algorithm proceeds to the **Generalize** step in Figure 4 below. If not, the algorithm takes the join of the strongest theories, T_0, selects a model for it and presents it to the user. If the user finds the model acceptable, we proceed again to **Generalize**. If not, the algorithm constructs all possible chains from each strongest theory to T_0.

In this case, we know that the theories the user desires are bounded by T_0 above and $Consistent_j$ below. We traverse each chain from the most general theories (top down) showing models for each module. If accepted, we terminate exploration of that chain and store T_{test} in $Proposed_j$. Otherwise, we iteratively remove the top element till there is only one element left in the chain. At this point, we test if the union of the selected models in each chain is consistent. If so, we have developed the strongest axioms from this hierarchy for the user, otherwise the algorithm terminates. We then move to the component of the algorithm for combining results from the various hierarchies.

If the resultant theories are not mutually consistent, the algorithm terminates. The reason for this apparent inconsistency in the user responses may be that the user is trying to define a relation over multiple sorts simultaneously. Alternatively, they may desire a theory not in the repository. Since these cases cannot be disambiguated, the algorithm simply exits.

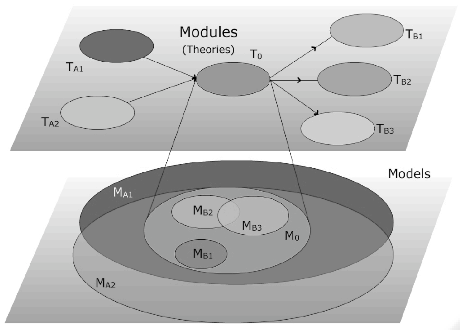

Figure 3: The relation between modules (theories) and their associated sets of models. Moving rightwards, we have non-conservative extensions.

In the case where $Consistent_j$ consisted of a unique theory in a core hierarchy, then instead of the chain method, we look directly at the theories above and below the initialized theory. The hierarchy functions as a "map" which the algorithm navigates, propelled by user responses to proposed models. Imagine the user is situated at T_0 with models "above" and "below" as in Figure 3, then the following algorithm applies. To ensure that T_0 is the desired theory, we must exhaust the search space by constructing models for those above and combinations of those below. As before, to account for inadvertent properties in the set of user models, we need to test the weaker theories. We may do this by negating T_0 and constructing a model for each T_{Ai}. If any of these models are accepted, we know that a more general theory is required, and we reset T_0 to one of those above.

If we can no longer generalize, then we look to prune those below T_0. We may construct as many models corresponding to those areas in M_{T_O} above. In this case, we learn through rejection. If these models are accepted, then the proposed axioms do not change. A rejection however, results in the addition of $\neg T_{Bi}$ to the proposed axioms. In this process, as illustrated in the steps below, we may ensure that the strongest axioms from core hierarchy j are selected. These steps are reflected in Figure 4.

Up to the **Combine Hierarchies** steps, we have selected the strongest theories for the user's intended models from a single hierarchy. Of course, it may have been the case that the user models satisfied theories in more than one hierarchy. The algorithm up to this point repeats for each core hierarchy which had at least one theory which were satisfied by all the user models, now we must combine the results.

For each core hierarchy j, which provided full cover the user models, we now have the set $H_{0,j}$ which are the strongest theories. We recall that our initial stipulation was that each Core Hierarchy not share a non-conservative extension with another. Consequently, we need only check if the union of all $H_{0,j}$ is consistent. If so, we have determined the strongest axioms that are consistent with all user models, all those denoted accepted and inconsistent which those marked reject. Otherwise, the algorithm terminates due to possibly conflicting inputs by the user.

4.4 Theorem and Correctness Proof

Given user inputs and a modular repository, the correctness of the algorithm is characterized by the following theorem:

Theorem 1. *The algorithm* Axiom-Generation *generates a set of the strongest theories H_0 that satisfy the following properties:*

1. *H_0 is a composite of the theories from the ontology repository*
2. *H_0 is consistent with all user generated models.*
3. *H_0 is consistent with all selected models that the user denoted* Accept
4. *H_0 is inconsistent with all selected models that the user denoted* Reject

The full correctness is too long for this paper, here we only provide a sketch. We first need to prove four lemmata, the first (lemma 1) showing that the algorithm applies i.e. ensuring that there is at least one theorem in some core hierarchy that satisfies all the user models. If such a theory exists, then the algorithm may continue, otherwise it terminates as shown in Figure 2 above. Lemma 2 establishes that for each core hierarchy, the algorithm initializes at the strongest theories which are satisfied by *all* the user models. Briefly, using lemma 1, we can show this is achieved by the *Rid Parents* sub process.

Lemma 3 shows that for each investigable core hierarchy, we may attain a set of theories $H_{0,j}$ which are

```
1. For every hierarchy, where $|Consistent_j| > 0$
2. If $|Consistent_j| > 1$ and user rejects $M_{To}$ (where $M_{To}$ is a model selected for $T_0 = \bigvee(CH_{m,j} \in Consistent_j)$)
   (a) Construct all chains, Chain($T_0$, A), where $A \in Consistent_j$
   (b) For each chain, create $M_{Ttop}$ for $T_{top}$ (top of chain k)
      i. If $M_{Ttop}$ is accepted, $M_{Ttop} \in Accept_j$, end chain, add $T_{top}$ to $Propose_j$
      ii. If $M_{Ttop}$ is rejected, $M_{Ttop} \in Reject_j$, remove top element of chain, $T_{top}$ is now new top element of chain. If top=bottom, end chain, add $T_{top}$ to $Propose_j$
   (c) If no more chains
      i. If $Propose_j$ is not consistent **End Algorithm**
      ii. If $Propose_j$ is consistent, check next hierarchy, otherwise, goto **Combine Hierarchies**
   **Generalize**
3. Set $T_0 = Consistent_j$
4. For each $A_p \in Above_{To}$, Construct $M_{Ap}$ from $A_p \bigcup \neg Below_{Ap}$
   (a) If $M_{Ap}$ is rejected, then remove $A_p$ from $Above_{To}$ and $M_{Ap} \in Reject_j$
   (b) If $M_{Ap}$ is accepted, then $M_{Ap} \in Accept_j$ and set $T_0 = A_p$, generate new $Above_{To}$
   **Specialize**
5. Construct $M_{To}*$ from $T_0* = T_0 \bigcup \neg Below_{To}$
   (a) If accepted, $M_{To*} \in Accept_j$
   (b) If rejected, $M_{To*} \in Below_j$ and $Propose_j = Below_{To}$
6. For every $B_p \in Below_{TO}$ Construct $M_{Bp}*$ where $B_p* = B_p \bigcup \neg(Below_{To}/B_p)$
   (a) If accepted, $M_{Bp}* \in Accept_j$ try next $B_p$
   (b) If rejected, $M_{Bp}* \in Reject_j$, add $\neg B_p$ to $Propose_j$
7. $H_{0,j} = \bigcup Propose_j$
8. **Combine Hierarchies**
   (a) Let $Combine = \bigcup_j H_{0,j}$
   (b) If $Combine$ is consistent, then display: "The strongest set of axioms from the repository which correspond to your inputs are $Combine$" **End Algorithm**
   (c) Else there is no set of axioms in the repository.
```

Figure 4: The Algorithm *Generalize-Models*.

the strongest theories in that hierarchy that are consistent with all the elements of $Accept_j$ and inconsistent with all elements of $Reject_j$. Finally, lemma 4 ensures that the set *Combine* is the union of the strongest theories from the repository that are consistent with *every* $Accept_j$ and inconsistent with every $Reject_j$.

With these lemmata, we may prove the correctness theorem, since by lemma 1 we know the algorithm engages only if there is a hierarchy in the repository which satisfies all the user models (hence satisfying property 1 and 2). Lemma 2 shows that we can always select the strongest theories in a core hierarchy, while lemma 3 shows that these theories will be consistent with $Accept_j$ and inconsistent with $Reject_j$. Finally, lemma 4 shows that we may extend this result to all *Accept* and *Reject* sets, otherwise the algorithm terminates. Combining these results satisfies the final two properties of the theorem.

5 USE CASE

In this section we will briefly show how this algorithm would apply for a real world concept. We take as given the existence of an ontology for partially ordered sets organized as a core hierarchy. We aim to add axioms to the *flows* relation from SUMO, which aside from categorization axioms, only asserts its anti-symmetry and transitivity (Nichols, 2004).

User models have been constructed by looking at a map and selecting rivers as nodes and edges as the *flows* relation. Figure 5 illustrates two such depictions of models for flows. Each model is converted into a complete diagram and tested against the poset hierarchy to see which theories are satisfied by them.

Axioms for posets, comparability graphs, down forests and down trees are all satisfied by these models. However we initialize only at down tree since all the other theories have children in $Consistent_j$. Since we have a unique theory as the current lower bound, we do not invoke the chain investigation and instead try to generalize and then specialize.

In this case, $Above_{downtree}$ = {Down Forest, Bounded Meet Semi Lattice}. The algorithm selects a model for each that is *not* a down tree. The first model proposed is a series of down trees (a down forest). As we are acting as the user in this case, our understanding is that most uses of flow have every river or body of water flowing into a unique body of water.[2] Hence, we reject this model.

The next model is that of a bounded meet semi lattice that is not a down tree (i.e. a child may have more

[2]not necessarily so, but assumed for this use case

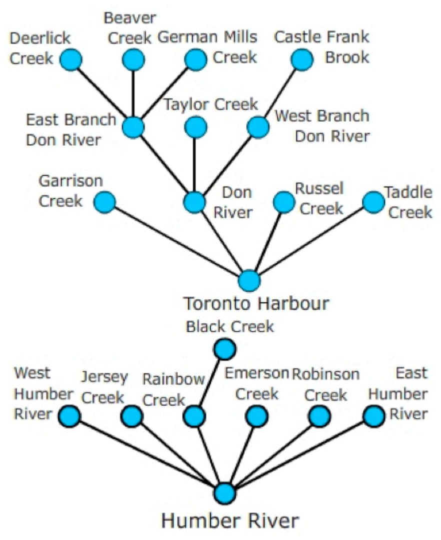

Figure 5: Hasse diagram conventions for two user models. Nodes are names of rivers, edges are the flows relation.

than one parent). This seems acceptable since rivers may bifurcate then join up again, so it is accepted. The algorithm now has a new T_0, namely, bounded meet semi lattice. We create a new $Above_{TO}$ = poset in this case and again, we the present the user with a model for a poset which is not a bounded meet semi lattice. This model is rejected because it admits rivers ultimately flowing into more than one body of water.

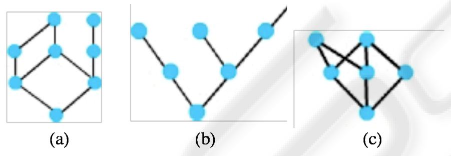

Figure 6: Several software proposed models: (a) uniquely a bounded meet semi lattice; (b) down tree; (c) bounded meet semi lattice that is not a down tree.

We now try to "Prune Specialize" by investigating $Below_{To}$ = {Down Tree, Meet Semi Distributive Lattice, Meet Pseudo-complemented Lattice, Meet Semi Modular Lattice}. The first model presented is thus a uniquely bounded meet semi lattice that does not exhibit the properties of any of its children. This seems plausible so we click accept. Next, the algorithm selects a model which satisfies each element of $Below_{To}$ exclusively. Every model seems like a potential models of rivers flowing into one another, so we accept each and they are all added to $Accept_j$. As there are no other combinations to try, nor other core hierarchies to investigate, we have $H_0 = H_{0,j} = Propose_j = \{$Bounded-Meet-Semi-Lattice$\}$ and $Accept_j$ is the set of all the user models plus those we accepted, while $Reject_j$ is the set of all the rejected models. We note that the axioms for bounded meet semi lattice were satisfied by all the models in $Accept_j$ and not by those in $Reject_j$. Thus we have provided axioms for *flows* by reusing axioms for *leq* as constrained by the theory for bounded meet semi lattices.

6 DISCUSSION

In this paper we have presented two basic design principles for any ontology repository which enables an ontology design algorithm. The algorithm uses the repository as a "map" of theories through which it navigates to identify the most appropriate characterization of the user's intended models.

It relies on the structure of the repository and the nature of non-conservative extensions. User models initialize the algorithm within one or more core hierarchies in a single abstraction layer. The algorithm then tries to determine whether there were accidental properties in the user models by exploring weaker, more general theories. It does so by proposing models which do not exhibit the stronger qualities. Once these have been exhausted, the algorithm attempts to identify the strongest possible theories, by exploring the set of modules below the selected theory. The algorithm uses falsification to drive its navigation of the repository. It selects models based on the interesting areas as shown in Figure 3. Each region in the "model space" is an interesting one which is accounted for and tested by the algorithm.

In its current guise, the algorithm only formalizes one relation at a time. A Sandbox Tool has been developed for graphs, which allows users to depict models for binary relations while explicitly specifying the translation conventions. Again, given adequate specification, a non-alpha numeric representation may be converted into a complete diagram for a model, in any sensory arrangement. The only restriction is that such depictions must have a clear, unambiguous and repeatable translation into complete diagrams. This process is suited for relations with arity of 2-5 - any higher and it becomes difficult to find a suitable representation for the models. Some relations with higher valence may be projected into a smaller dimension given a contextual framing.

The precision and accuracy of the proposed axioms of the algorithm are dependent on the breadth and depth of the reference repository. There may always be some additional axioms which the designer wishes but are not present in the repository. While the algorithm is necessarily silent about these axioms, should they be formalized, the repository can grow to accommodate these novel learnings.

The algorithm relies on a theorem prover and/or satisfiability checker to verify that a particular model indeed satisfies the axioms of an ontology. It is important to realize that the algorithm is not dynamically constructing new models of the axioms (which in general is undecidable). Since the repository is a (relatively) static entity and we know which theories are above and below any other theory, we can construct all the necessary models before engaging the algorithm. We need only convert the diagrams into the desired representation convention on the fly.

7 CONCLUSIONS

The work presented here only scratches the surface of a novel way of axiom generation - that of conversing at the semantic level. It greatly reduces the burden on subject matter experts to intimately learn a formal language. Moreover, it skirts many of the issues of Upper Ontologies by not taking positions on the nature of "reality," but by capturing the logical structures that underlie many concepts. Consequently, user concepts do not necessarily map into particular conceptualizations of space and time, but rather the more abstract concepts of "properties of binary (or higher arity) relations."

Moreover, as interrelations between layers of abstractions become formalized, more natural model representations might also be used. For example a molecular biologist might be able to draw models based on actual molecules instead of abstracted graphs. Similar to how users can customize *skins* for software graphical user interfaces, model representations may be *skinned*, so as to reflect a representation more natural to a subject matter expert's domain (i.e. a molecular biologist might look at spatial configurations of molecules.)

Closely related, are questions of representing high arity relations, or models of infinite size. Contexts and/or project may address the former, while ellipses and/or navigable fractals may be promising depictions. Explicit conventions would mitigate some of these issues, but these ideas are preliminary at the moment.

Similarly, we currently axiomatize structures that are isomorphic to the extensions of a single relation. Extending this work to multiple relations presents an interesting challenge. How does the interaction of relations affect which models to propose?

Lastly, an extension to the repository would see it retain the relation names when a user engages the algorithm to axiomatize a relation. In this way, we might be able to enrich the user browse experience by presenting different, previously user defined notions of say, *time*. One could then compare *time* as defined by A vs. B and see whether either of those definitions corresponds to the *time* they wish to axiomatize.

In this paper we have shown how the organization of theories within an ontology repository can be exploited to provide an axiomatization of a class of models. Furthermore, such a repository structure provides the foundation for many other applications in ontological engineering. One such example is an algorithm for the generation of semantic mappings for ontologies consistent with those in the repository - the repository serves as a central family of interlingua ontologies through which alignment may be achieved.

REFERENCES

Delugach, H. (2007). Common logic (cl) - a framework for a family of logic-based languages. Technical report, Geneva.

Guarino, N. (1998). Formal ontologies and information systems. In *Formal Ontology in Information Systems. Proceedings of FOIS'98*, pages 3–15. IOS Press.

Hou, C. J., Musen, M., and Noy, N. F. (May 2005). Ezpal: environment for composing constraint axioms by instantiating templates. *International Journal of Human-Computer Studies*, 62:578 596.

Luettich, K. and Mossakowski, T. (2004). Specification of ontologies in casl. In *Formal Ontology in Information Systems Proceedings of the Third International Conference (FOIS-2004)*, pages 140–150. IOS Press.

Nichols, D. (2004). Ontology of geography. *http://sigmakee.cvs.sourceforge.net/*checkout*/sigmakee/KBs/Geography.kif.*

Presutti, V. and Gangemi, A. (2008). Content ontology design patterns as practical building blocks for web ontologies. In *27th International Conference on Conceptual Modeling*, pages 128–141. Springer.

Staab, S. and Maedche, A. (2000). Ontology engineering beyond the modeling of concepts and relations. In *14th European Conference on Artificial Intelligence; Workshop on Applications of Ontologies and Problem-Solving Methods*.