Proof-Guided Ontology Development using Pattern Rules

Liwei Deng and **Alan Bundy** and **Fiona McNeill** and **Alan Smaill** [1]

Abstract. In this paper we present a proof-based method of developing ontologies that satisfy requirements, where the proof process guides the development of the ontology. We formally define our procedure and illustrate it with examples.

1 INTRODUCTION

Ontologies are (formal) models of some aspect of the real world, with applications such as information sharing, semantic search and interoperability. There are various ontologies in use, including many business and biomedical ones.

Developing (and maintaining) such ontologies involves many activities, sources of knowledge and people. The experience over the years have been analysed, organised and developed into *methodologies*, describing the whole process of developing an ontology; see [2] [1] for meta-surveys.

Generally there are three main stages:

1. first, purpose and requirements are identified, relevant knowledge gathered, and an informal ontology may be built;
2. then, the formal ontology is created;
3. finally, the ontology is evaluated against the purpose and requirements, modified accordingly, and maintained as long as necessary.

We aim to find good methods for stages 2 and 3 above, that of creating the formal ontology that satisfies formal requirements. Such ontologies have the quality of *relative correctness*, correct relative to the requirements.

Establishing the satisfaction of some formal requirements can take the form of proofs: if the (statements of the) formal ontology are seen as a set of axioms, these formal requirements are conjectures on the ontology to be proved.

For example, for an anatomy ontology, we could have a requirement

> The human body is left-right symmetric externally;

which can be formalised as

> *external_symmetric(human_body)*.

We will be able to deduce this from the ontology if, for example, the ontology had, for each external component, both a left part and a right one, or is designated self-symmetric.

This approach of formally proving satisfaction of requirements by ontologies was first introduced in TOVE (TOronto Virtual Enterprise) [3], and we will discuss this later in Section 3, Related Work.

[1] University of Edinburgh, Scotland, email: {L.Deng-2@sms.ed.ac.uk, bundy@staffmail.ed.ac.uk, f.j.mcneill@ed.ac.uk, A.Smaill@ed.ac.uk}

In this paper we are only interested in formal requirements that could be stated as conjectures on and proved to follow from the formal ontology and henceforth we will be using the term *formal requirements* to mean only these ones.

For creating formal ontologies where formal requirements could be proved, we propose *Proof-Guided Ontology Development*, where we use the formal requirements to *guide* the development of the ontology. We start with the formal requirements and use their proof attempts to guide the user on the relevant classes, relations and rules to add to the ontology so that it reflects the user's view of the domain, at the same time as giving proofs of the statements. When the development of the ontology is finished, all the requirements should be proved, so we know that they are all satisfied.

This use of formal requirements to guide the development of the ontology is different from existing approaches.

The rest of the paper is structured as follows: in Section 2 we show the use of certain *pattern rules* in guiding the development of the ontology, starting with a simple example in Section 2.1, followed by a formal description of our procedure as an algorithm in Section 2.2, which is then applied to a more complicated example in Section 2.3, then a brief discussion in Section 2.4, and a brief description of the implementation of the example in Section 2.5, followed by its evaluation in Section 2.6; we then discuss related work in Section 3, described on-going and planned work in Section 4, before concluding in Section 5.

2 USING PATTERN RULES

Our development process aims to end in an ontology where all the formal requirements are satisfied, and to get there we aim to go through an iterative series of states in each of which an intermediate ontology satisfies an intermediate set of formal requirements (which could be a subset of the final ones).

We start with a state with no formal requirements and an empty ontology (with no classes, relations, properties, etc.), so that trivially the ontology satisfies the requirements. The user who is building the ontology then changes the state by adding formal requirements, and classes, relations and properties to the ontology. The resulting intermediate ontology is checked to see if it satisfies the requirements. If not, then we aim to use the formal requirements to guide the user to make changes to the current ontology to arrive at a state where the requirements are all satisfied (again). We carry on like this until we get to the ontology we want.

Note from the above description that we are not starting from complete scratch, but assume that the user has some idea of the requirements, and hence classes, relations and axioms, for the ontology and will be providing them.

A key part of this process is the guidance we provide to the user in cases the intermediate ontology does not satisfy the requirements.

We said above that we will use the formal requirements to provide guidance, but on their own usually they will not be enough. In this paper we will show the use of certain pattern rules as rules of implication in giving guidance to the user.

We first illustrate the idea with a simple example.

2.1 A Simple Example

Our simple example is that of a car ontology, where we are trying to model the structure of a car. We start with no formal requirements and an empty ontology, as described above. The user then adds the requirement

The engine is connected to the wheels;

which could be formalised as

connected(engine, wheels).

Now if the requirement is added to the ontology, we would have immediate (and trivial) satisfaction of the requirement by the ontology, but there are various reasons for not doing this. One is that we want to use the requirement to guide the development of the details of the ontology, in a specific way, which we will see below soon. Another is that eventually we want to prove generalised versions of the above requirement and these cannot be axioms of the ontology.

Instead, the above requirement would simply imply the addition of the classes *engine*, *wheels* and the relation *connected* to the ontology.

To these, the user might further add the axioms

connected(engine, axle), connected(axle, wheels)

to the ontology to model how the engine is connected to the wheels.

Now a check can be performed to see if the current ontology satisfies the above requirement. It is clear that we will not be able to deduce it: it is not an axiom of the ontology, but it cannot be implied by the other axioms either as we do not have any rules to use for implication. Now we note that if we had properties for the relation *connected* such as transitivity, which could be formalised as

$$connected(X, Z),\ connected(Z, Y) \rightarrow connected(X, Y), \quad (1)$$

for example, then it could be used as a rule of implication, with the two atoms on the left implying the one on the right.

So suppose we can have such properties, the question is then which ones should we use? A natural first try is the commonly occurring ones, such as transitivity, symmetry, etc. But where do we get such properties from if they are not there?

One possibility is to have a generic versions of these commonly occurring properties, which could be instantiated with any particular relation we might want. For transitivity such a generic property could be formalised as follows

$$P(X, Z),\ P(Z, Y) \rightarrow P(X, Y), \quad (2)$$

with P a variable over relations. If P were instantiated to *connected*, we would get (1) above. In this way, we get rules to try in a proof attempt of the formal requirement; these are our *pattern rules*. A list of some commonly occurring generic properties as pattern rules is shown in Figure 1; this is not intended to be exhaustive.

Our guidance then proceeds by backward reasoning. We start our attempted proof of the formal requirement

connected(engine, wheels)

as the goal by first noting that it is not an axiom of the ontology, so will need to be deduced. Then we look for a rule that could be used to break down the goal into subgoals. We note that properties such as (1) could be used as rules, and first look for a rule of this form, but find none. So next we look for a generic rule of the form (2) to obtain an instantiated rule of form (1), where the relation variable in (2) is instantiated to the relation in our goal. So from (2) we obtain (1); the conclusion of this implication is removed by resolution leaving only the two premises,

$$connected(engine, Z),\ connected(Z, wheels) \quad (3)$$

as subgoals. We note that they could be unified with the axioms in the ontology, thus giving a derivation of our original goal. This triggers a notification to the user of a potential derivation of our goal, depending on the use of the instantiated rule (1), which could be a property for the relation *connected*. The user is asked if (1) is indeed a property for the relation, with the successful derivation providing evidence in its favour. If the user accepts, the property is added to the ontology, and we have guided the user to a state where all formal requirements could be derived.

Having given the example, we can now state our hypothesis for this paper:

Proof-Guided Ontology Development using pattern rules can be used to provide guidance in the development of real ontologies.

We will come back to the hypothesis when we discuss evaluation in Section 2.6, but next we present a formalisation of the above procedure.

2.2 The General Algorithm

Our procedure takes formal requirements and applies them to ontologies using pattern rules, to provide guidance. So before presenting a formalisation of this procedure as an algorithm, we first give formal definitions for each of the three involved elements. Our formalisation is in the style of logic programming, see e.g. [7].

2.2.1 Requirements and Axioms

We first give a basic definition, that of *atoms*, that will allow us to define formal requirements and atomic axioms.

Definition 1 (Atoms) *An* atom *is a formula $P(T_1, \ldots, T_n)$, where P is a predicate and the T_is are terms, which can only be constants or variables. If P is a constant then $P(T_1, \ldots, T_n)$ is a* regular *atom; if the T_is are also constant then it is a* ground *atom. If P is a variable then $P(T_1, \ldots, T_n)$ is a* pattern *atom.*

Remark 1 *The restriction of terms to constants or variables in the above definition has implications for the size of the search space, see Remark 4.*

Example 1 (Atoms) *connected(X, Y) and connected(engine, Y) are both regular atoms, where connected is the predicate in both cases, with X and Y the terms for the first atom, $engine$ and Y the ones for the second.*

We can now define .

Definition 2 (Formal Requirements) *A* formal requirement *is a ground atom, where the predicate is a relation and the terms are classes of the ontology.*

Definition 3 (Atomic Axioms) *An* atomic axiom *is a ground atom, where the predicate is a relation and the terms are classes of the ontology.*

Remark 2 *For this paper formal requirements and atomic axioms have the same definition, but in future work this is unlikely to be the case, as we expect to have e.g. more general requirements (cf discussion about why requirements are not just added to ontologies as axioms at the start of Section 2.1).*

Example 2 (Formal Requirements, Atomic Axioms) $connected(engine,\ wheels)$ *is a formal requirement;* $has\_part(wheels,\ frontwheels)$ *is an atomic axiom (these are taken from our example ontology in Section 2.3 that we will see later).*

2.2.2 Pattern Rules

Having defined atoms, we can use them to define (Horn) clauses of various types, including pattern rules.

Definition 4 (Horn Clauses) *A* Horn clause *is either a rule clause or a goal clause. A* rule clause *is a formula* $Body \implies Head$, *where* $Head$ *is an atom and* $Body$ *is a (possibly empty) set of atoms, where all the terms in each atom are variables. A* goal clause *is a set of atoms. If the goal clause is the empty set it is called the* empty clause.

A rule clause is a regular clause *iff it contains only regular atoms; it is a* pattern clause *if it contains pattern atoms.*

A regular rule clause where the predicate is a relation of the ontology is a property of the relation.

Remark 3 *We could have said* first order *instead of* regular *and* second order *instead of* pattern *in the above definition, but we wanted to emphasise the notion of pattern rules.*

Definition 5 (Pattern Rules) *A* pattern rule *is a pattern clause with only one predicate variable.*

Example 3 (Horn Clauses)

$$connected(X, Z),\ connected(Z, Y) \rightarrow connected(X, Y)$$

is a regular clause; it is also a property of the relation connected *if* connected *is a relation of an ontology;*

$$P(X, Z),\ P(Z, Y) \rightarrow P(X, Y)$$

is a pattern rule (pattern clause), as is

$$connected(X, Z),\ P(Z, Y) \rightarrow P(X, Y);$$

$\{connected(X,\ Y)\}$ *and* $\{connected(engine,\ Y)\}$ *are goal clauses.*

Definition 6 (Ontologies) *An* ontology *contains (non-exhaustively) a set of classes, relations, atomic axioms and regular rules.*

Example 4 (Ontologies) *See Figure 2.*

2.2.3 Proof Guidance Algorithm

Next, we use the various types of clauses to define *search spaces*, and finally our formal algorithm, the *Proof Guidance Algorithm*.

Definition 7 (Search Spaces) *A* search space *is an OR-tree*[2] *in which the nodes are labelled by goal clauses and the arcs are labelled by rule clauses. Exactly one of the atoms of each goal clause is called its* selected goal.

Suppose $\{A_1, A_2, \ldots, A_n\}$ *is the goal clause labelling a node with* A_1 *the selected goal, and the rule clause* $Body \implies Head$ *labels one of the arcs descending from this node. Then the daughter node at the other end of this arc is labelled with* $Body\sigma \cup \{A_2, \ldots, A_n\}\sigma$, *where* σ *is the unique most general unifier of* A_1 *and* $Head$.

A proof *of the goal clause labelling the root of a search space corresponds to a branch in which the leaf node is the empty clause. A* regular proof *is one that only uses regular rules. A* pattern proof *is one that uses at least one pattern rule.*

Example 5 (Search Spaces) *If* $\{connected(engine, wheels)\}$ *is the goal clause labelling a node and the rule clause*

$$connected(X, Z),\ connected(Z, Y) \rightarrow connected(X, Y)$$

labels one of the arcs descending from this node, then the daughter node at the other end of this arc is labelled with $\{connected(engine, Z),\ connected(Z, wheels)\}$.

Definition 8 (Proof Guidance Algorithm) *Given a formal requirement, (ground regular goal clause with one atom), a set of regular rules and a set of pattern rules, the search space is grown depth first in two passes. On the first pass only the regular rules are used. On the second pass, both regular and pattern rules are used. Thus only regular proofs will be found on the first pass and only pattern proofs on the second pass.*

If a pattern proof is found, then the pattern rules used in the proof will have their predicates instantiated to the one in the unifier; these instantiated pattern rules *are presented to the user to ask if they are all true.*

Example 6 (Proof Guidance Algorithm) *If*

$$P(X, Z), P(Z, Y) \rightarrow P(X, Y)$$

is a pattern rule used in the proof and the unifier is $connected$ *then the instantiated pattern rule is*

$$connected(X, Z),\ connected(Z, Y) \rightarrow connected(X, Y).$$

Remark 4 *The restriction of terms to constants or variables in the definition of atoms (Definition 1) means that if the predicate and terms of an atom in a goal clause are instantiated to relations and classes of a finite ontology, there are only finitely many possibilities; so the search space is finite.*

The algorithm does not specify how it is to be applied to ontologies, e.g. how goal atoms are instantiated; this is a deliberate design decision to make it general, so that different search strategies can be applied on top of it; we will see an example in the next section.

[2] OR-trees contrast with AND-trees. In an OR-tree, you can choose one branch and stick with it; in an AND-tree, all branches must be solved. The analogy is with goals of the form $P \vee Q$ versus $P \wedge Q$. Since we will be using depth first search (see Definition 8), our search space is an OR-tree.

2.3 A More Complicated Example

Having presented our algorithm formally, we now apply it to a more complicated version of the previous example of a car ontology; although it is small it is non-trivial.

2.3.1 Pattern Rules for Our Example

As before, we start with no formal requirements and an empty ontology, but this time with a list of generic properties as pattern rules, shown in Figure 1.

This is not meant to be an exhaustive list, but a list of what we regard as among the most commonly occurring properties. Apart from the first, the other three are composed of pattern atoms only and have their familiar meanings. As for the first, we called it *upward inheritance* as it describes the situation where a relation that holds between two classes Z and Y is inherited upwards along hierarchies of *has_part* for one of the classes, Y, so that it also holds between Z and X, the superclass of Y. We singled out inheritance along hierarchies of *has_part*, rather than a generic relation, as we feel that this is a very common situation, making this generic property a mixture of pattern and regular atoms. We further note that the relation that is inherited upwards must not be *has_part*, so that if it is instantiated then it will have two relations, unlike the other three.

These properties are not requirements to be proved and are not part of any ontology, so we still have the (trivial) satisfaction of the requirements.

Type	Generic Property
upward inheritance	$has\_part(X,Y), P(Z,Y) \rightarrow P(Z,X)$
transitivity	$P(X,Z), P(Z,Y) \rightarrow P(X,Y)$
symmetry	$P(Y,X) \rightarrow P(X,Y)$
reflexivity	$P(X,X)$

Figure 1. A list of generic properties.

2.3.2 Starting the Development Process

The user developing the ontology then adds the requirement

The engine is connected to the wheels;

formalised as

connected(engine, wheels)

and some axioms to the ontology; the classes and relations in these are also added to the ontology, which are shown in Figure 2.

We first check that the ontology is consistent. In this case, our atomic axioms are clauses with no negation, so they are consistent. In general, this would involve searching for the empty clause; if we fail then the ontology is consistent. The restriction of terms to constants or variables in the definition of atoms (Definition 1) means the search will terminate, so we will a definite answer.

Having done this, we next check (or the user could initiate the check) if the requirement can be derived from the ontology; we can do this by applying the Proof Guidance Algorithm, since the first pass using only regular rules does exactly this, and has termination. In this case, we have the answer that it cannot be derived. Therefore we use the (second pass of the) Proof Guidance Algorithm to guide the user on the additions to the ontology to make it satisfy the requirement.

Classes	Relations	Atomic Axioms
car	has_part	has_part(car, engine) has_part(car, wheels)
engine axle	connected	connected(engine, axle) connected(axle, leftfrontwheel)
wheels frontwheels leftfrontwheel		has_part(wheels, frontwheels) has_part(frontwheels, leftfrontwheel)

Figure 2. The more complicated car ontology example.

2.3.3 Heuristics for the Algorithm

We have already noted that the algorithm does not specify how it is to be applied to ontologies; to do so we use the following heuristics.

Definition 9 (One Atom Heuristic) *If $\{A_1, A_2, \ldots, A_n\}$ is not the formal requirement (initial goal clause), we require the existence of a unifier σ such that all but at most one of $\{A_1, A_2, \ldots, A_n\}\sigma$ are axioms of the ontology; these are then removed from the goal clause, leaving a goal clause with at most one atom; otherwise the node is a leaf node.*

Example 7 (One Atom Heuristic) *For example, if*

$$\{connected(engine,\ Z),\ connected(Z,\ wheels)\}$$

is the goal clause labelling a node, then Z could be unified with (instantiated to) $axle$ to make the first of the two goals an axiom of the ontology (in Figure 2), which is then removed from the goal clause, leaving

$$\{connected(axle,\ wheels)\};$$

on the other hand if

$$\{connected(leftfrontwheel,\ Z),\ connected(Z,\ wheels)\}$$

is the goal clause labelling a node, then we do not have any unifiers which could make one of the goals in the clause an axiom of the ontology, so the node is a non-empty leaf node (hence does not give a proof).

The idea here is to use the ontology to cut down the search space, hopefully in a way that directs the search towards a proof. If we start with a goal of one atom then when we grow the search space using the Proof Guidance Algorithm then we are unifying the atom with rule clauses to produce goal clauses, usually of more than one atom. Now, if all the atoms in one such goal clause were unifiable with axioms in the ontology, then we would have a proof of the original goal, as in the simple example in Section 2.1.

However, this will rarely be the case; so for our first heuristic, we aim for the next best thing, trying to unify all except at most one of the atoms. This leaves a subgoal of at most one atom to be proved, for which we again attempt to reduce to a subgoal of at most one atom, and so on, until we find a proof or fail.

This heuristic hugely cuts down the search space, though we are aware that it sometimes prevents proofs from being found (see discussion in Section 4).

Definition 10 (Instantiated Pattern Rule Heuristic) *In the second pass of the Proof Guidance Algorithm, instead of using the patterns rules directly, we will have their predicates instantiated to the predicate in the atom of the formal requirement (initial goal clause) and use these* instantiated pattern rules *(the same kind as the ones in Definition 8).*

We also split the second pass into subpasses. On the first subpass only one pattern rule is used; on the next subpass we can use one more pattern rule, and so on, until there are no more pattern rules.

With the Proof Guidance Algorithm, the pattern rules labelling the arcs leading from the formal requirement (initial goal) effectively have their predicates unified with (instantiated to) the one in the requirement, so this is minimal in the number of pattern rules used if an order is imposed on the pattern rules (as is indeed the case in our implementation, see Section 2.5), with the effect on goal clauses lower down the search tree.

By splitting the second pass into subpasses we make our search space small at the start and only grow it if no proof is found.

We can now give a summary of the running of the algorithm with the heuristics. As before, the search space is grown depth first in two passes. On the first pass only the regular rules are used. Each time a new node is grown, its goal clause is reduced using the *One Atom Heuristic* to give either

1. an empty clause, which shows a proof of the original goal has been found;
2. a clause with one atom, which is a leaf node if it has already appeared;
3. a clause with more than one atom, which is a leaf node.

The search space is finite If a proof is not found on the first pass, we go to the second pass, which is split into subpasses. On the first subpass, we obtain an instantiated pattern rule as per the *Instantiated Pattern Rule Heuristic*, which together with the regular rules is used to grow the search space, and then proceed as on the first pass.

If a proof is not found, we go to the next subpass, where we obtain one more instantiated pattern rule, and carry on as before. This goes on until either we find a proof or run out of pattern rules to instantiate.

2.3.4 The Running of the Algorithm

Given these two heuristic we can now give an overview of the running of our algorithm on the car ontology in Figure 2.

We are trying to guide the user on the additions to the ontology to make it satisfy the requirement

connected(engine, wheels),

which is our initial goal.

The first pass using only regular rules is just the check on the derivation of the requirement from the ontology from earlier, which fails, so we need the second pass.

On the second pass, without loss of generality the first subpass uses the first pattern rule from Figure 1,

$$has\_part(X, Y), P(Z, Y) \rightarrow P(Z, X),$$

the upward inheritance property, which we instantiate to

$$has\_part(X, Y), connected(Z, Y) \rightarrow connected(Z, X)$$

using the Instantiated Pattern Rule Heuristic, giving us the upward inheritance rule for *connected*, then unify with the requirement to give

$$has\_part(wheels,\ Y), connected(engine,\ Y)$$

$$\rightarrow connected(engine,\ wheels). \quad (4)$$

The two terms on the right hand side are the subgoals of the initial goal, proof of which will give us the proof of the requirement. We now apply the One Atom Heuristic to the subgoals and note that we can instantiate Y to $frontwheels$ to make the first of the two goals an axiom of the ontology, leaving us to prove

connected(engine, frontwheels).

One further application of the above steps reduces us to proving *connected(engine, leftfrontwheel).*

But now it is easy to see that with only this rule we will run out of possibilities quickly; further unifications with the upward inheritance rule will bring goals that could not be unified with axioms in the ontology.

So we go to the second subpass, and get another instantiated pattern rule

$$connected(X, Z), connected(Z, Y) \rightarrow connected(X, Y), \quad (5)$$

the transitivity rule for *connected*. Applying this to *connected(engine, leftfrontwheel)* gives us

$$connected(engine, Z), connected(Z, leftfrontwheel), \quad (6)$$

and instantiating Z to $axle$ makes both axioms of the ontology, thus completes the proof.

We then present the two instantiated pattern rules used in the proof to the user for approval.

2.4 Discussion

In this paper, to prove our formal requirements we needed to come up with rules of implication which are also a properties of relations. We believe it is advantageous for an algorithm running on a computer to do this job, for the following reasons:

1. arguably, it is easier for a human to understand such rules than to design them, particularly as they have formal logical formulations such as (1);
2. a relation could have many properties, designing them is a diversion from the modelling of the domain, so getting a program to suggest them is a good division of labour;
3. many relations could have similar properties to be added, so together with the above two points this makes the task tedious and repetitive for a human, but computers are well suited to such tasks and can help to reduce the workload.

However, it is outside the scope of this paper to investigate these claims, empirically or otherwise.

2.5 Implementation

The Proof Guidance Algorithm, the two heuristics and example ontology have been implemented in a Prolog [7] program. One of the advantages of using Prolog is that it allows easy meta-programming; and indeed our programs are written as meta-interpreters. Additionally, Prolog's declarative style and logic-based semantics allow it to be used as a ontology language.

Prolog imposes orders on goals and rules, so the sets of atoms in goal clauses become lists (see Definition 4), as do the sets of regular and pattern rules (see Definition 8).

Implementation Note: So that pattern atoms can be unified with other atoms using only first-order unification, an atom $P(T_1, \ldots, T_n)$ is represented internally as $atom(P, T_1, \ldots, T_n)$.

This means

connected(engine, axle)

becomes

atom(connected, engine, axle).

The command

suggest(atom(connected, engine, wheels), Proof, GoalList)

is used to try to prove

connected(engine, wheels)

(here *Proof* is the variable holding the final proof obtained, and *GoalList* holds the current list of goals to catch looping). The command succeeds, with the proof printed out at the end.

Proof = atom(connected, engine, wheels) :-
(atom(has_part, wheels, frontwheels) :- True), (atom(connected, engine, frontwheels) :-
(atom(has_part, frontwheels, leftfrontwheel) :- True), (atom(connected, engine, leftfrontwheel) :-
(atom(connected, engine, axle) :- True), (atom(connected, axle, leftfrontwheel) :- True)))

2.6 Evaluation

Our hypothesis for this paper is as follows:

> *Proof-Guided Ontology Development using pattern rules can be used to provide guidance in the development of real ontologies.*

The evaluation of our hypothesis is on-going, as we are looking for example ontologies where our method is applicable.

One interesting example we found is the OWL-Time Ontology [5]. This ontology has a relation *before*, which is defined to be anti-reflexive, anti-symmetric, and transitive on intervals. For example, the anti-reflexive axiom is given as

$$before(T_1, T_2) \rightarrow T_1 \neq T_2.$$

Given the axioms in OWL-Time, the following is a theorem of the ontology:

$$begins(t_1, T) \;\&\; ends(t_2, T) \;\&\; before(t_1, t_2) \rightarrow ProperInterval(T).$$

This says that if an interval T begins at instant t_1 and ends at instant t_2 and such that t_1 is before t_2, then it is a proper interval, which is itself defined as

$$(\forall T)(ProperInterval(T) \leftrightarrow interval(T)$$

$$\&\; (\forall t_1, t_2)(begins(t_1, T) \;\&\; ends(t_2, T) \rightarrow t_1 \neq t_2)).$$

This theorem is mentioned in the predecessor ontology of the OWL-Time Ontology, the DAML-Time Ontology [4], which has the same basic axioms and same axioms on the relation *before* as OWL-Time.

To prove this theorem, it seems to us that the anti-reflexive axiom on intervals for the relation *before* is needed, though the proof we could see is somewhat complicated.

OWL-Time does not have an anti-reflexive axiom on *instants* for the relation *before*,

$$before(t_1, t_2) \rightarrow t_1 \neq t_2;$$

if it did, the proof of the theorem would be much easier.

We would like to use our method of using a proof attempt of the theorem to suggest the anti-reflexive axiom on instants for *before*; but as it is, we cannot as we are not yet able to deal with quantification and negation. Assuming that we can, and that we have a pattern rule for anti-reflexivity, then it should be quite straightforward to make the suggestion.

We are currently working to extend our method to deal with quantification and negation.

3 RELATED WORK

In terms of overall approach to ontology development, TOVE (TOronto Virtual Enterprise) [3] is the most similar to us, in fact it is TOVE who first introduced the idea of formally proving satisfaction of requirements by ontologies.

In TOVE, requirements are called *competency questions*, a term and notion taken up by several subsequent methodologies.

Competency questions arise and are used in the following way. First, usage scenarios for the proposed ontology are identified; then from these a set of natural language questions, called competency questions, are derived. These are questions that the ontology should be able to answer given the usage scenarios.

These questions and their answers are then used to extract the main concepts, their properties, relations and axioms of the ontology, which are then used in the development of the latter.

Usually, the finished ontology is evaluated to check that it is able to answer the competency questions. In TOVE, this is done formally, with formalised competency questions (which are in the form of logical statements) proved to follow from the (formal) ontology; if the ontology is seen as a set of axioms, the formalised competency questions conjectures on the ontology to be deduced.

The difference between TOVE and our approach is that even though TOVE also intends that the competency questions are used to develop the ontology (axiomatize the ontology) in an iterative way, it is vague about how this is actually to be done. In fact, it [3] states that:

> *There may be many different ways to axiomatize an ontology, but the formal competency questions are not generating these axioms.*

In contrast, we use (the proof attempt) of formal requirements to (guide the) generation of the axiomatization the ontology, and gave an example of a specific way of doing this.

In terms of our example, of the specific way of (guiding the) generating the axiomatization the ontology, [6] is the most similar to us. The authors also propose a way of adding properties such as transitivity or symmetry to relations in ontologies, including automatically generated ontologies such as DBPedpia where this information tends to be lacking, though their method is rather different.

In [6], parts of ontologies of interest are matched with Ontology Design Patterns which already contains the properties, using an algorithm that takes into account both structural and lexical information, and which is configurable by the user. These Ontology Design Patterns are analogous to our Pattern Rules, in that they are also generic versions of commonly occurring constructions in ontologies, where these constructions are collections of related classes, relations and axioms. The algorithm first finds parts of ontologies that structurally match Ontology Design Patterns, e.g. the hierarchies match, then use lexical information such as synonyms to confirm the match. When a match is found, the properties in the Ontology Design Patterns are added to the matched items in the ontology.

In [6] the authors presented a feasibility study though there is no evaluation.

4 CURRENT AND FUTURE WORK

Work is on-going on the usage of pattern rules to provide guidance for ontology development, and we have further work planned.

4.1 Providing Guidance

On the basis of existing definitions for formal requirements, atomic axioms and pattern rules, we are developing more examples of requirements, axioms and more pattern rules, and the associated heuristics needed to deal with these.

We already have an example ontology, which is a more complicated version of the car ontology presented in this paper, for which the current heuristics are insufficient, in that they prevent proofs from being found. For this we have developed and implemented new heuristics.

We are also working on and planning for more general definitions of formal requirements and atomic axioms, for example variables in formal requirements, and the ability to deal with quantification and negation, which will allow us to deal with the OWL-Time Ontology example in full. For these more complicated ontologies heuristics may be insufficient and we will consider other measures.

In all cases, we will be testing our method on 3rd party ontologies.

4.2 Other Aspects of Ontology Development

In this paper we focused on providing guidance for ontology development, and did not consider or simplified other aspects of ontology development. We will consider these aspects in future, including:

- reuse of existing ontologies to provide requirements and axioms, and the issues this brings, such as dealing with equivalence of requirements and axioms;
- the ontology language used and how existing languages like OWL and CL could be used with our approach;
- use of existing tools for e.g. checking consistency.

We have discussed our proposal with a few ontology designers, but we are not currently working with any; this is something we will consider in future.

5 CONCLUSION

In this paper, we introduced the notion of *Proof-Guided Ontology Development*, and the use of pattern rules to guide the development of ontologies. We gave formal definitions of the notions involved, including formal requirements, atomic axioms and pattern rules, and gave a formal description of our algorithm. Our procedure is illustrated with an example, which has been implemented in Prolog programs.

We presented a hypothesis for this paper, the evaluation of which is on-going. We gave the example of the OWL-Time Ontology, to which our method does not yet apply, but noted the problems to be dealt with. These problems are to do with aspects of First Order Logic; though they would bring difficulties it seems reasonable that they could be dealt with. Assuming that they are, we noted that our method could potentially be applied.

Overall, we expect that our method is definitely useful for ontology development, and we are working to extend and evaluate it.

ACKNOWLEDGEMENTS

We would like to thank the referees for their comments which helped improve this paper.

REFERENCES

[1] M. Bergman. A brief survey of ontology development methodologies. online, August 2010. url: http://www.mkbergman.com/906/a-brief-survey-of-ontology-development-methodologies/, last retrived 28 Feb 2013.

[2] N. Dahlem, J. Guo, A. Hahn, and M. Reinelt, 'Towards an user-friendly ontology design methodology.', in *Interoperability for Enterprise Software and Applications*, pp. 180–186. IEEE Computer Society, (2009).

[3] M. Grüninger and M. Fox, 'Methodology for the design and evaluation of ontologies', in *IJCAI'95, Workshop on Basic Ontological Issues in Knowledge Sharing*, (1995).

[4] Jerry R. Hobbs. A DAML ontology of time, 2002.

[5] Jerry R. Hobbs and Feng Pan, 'An ontology of time for the semantic web', *ACM Transactions on Asian Language Processing*, **3**(1), 66–85, (2004).

[6] Nadejda Nikitina, Sebastian Rudolph, and Sebastian Blohm, 'Refining ontologies by pattern-based completion', in *Proceedings of the Workshop on Ontology Patterns (WOP 2009), collocated with the 8th International Semantic Web Conference (ISWC-2009), Washington D.C., USA, 25 October, 2009*, volume 516, (2009).

[7] L. Sterling and E. Shapiro, *The Art of Prolog*, MIT Press, Cambridge, MA, second edn., 1994.