The University of Manchester Research

Making Entailment Set Changes Explicit Improves the Understanding of Consequences of Ontology Authoring Actions

DOI:
10.1007/978-3-319-49004-5_28

Document Version
Accepted author manuscript

Link to publication record in Manchester Research Explorer

Citation for published version (APA):
Matentzoglu, N., Vigo, M., Jay, C., & Stevens, R. (2016). Making Entailment Set Changes Explicit Improves the Understanding of Consequences of Ontology Authoring Actions. In *20th International Conference on Knowledge Engineering and Knowledge Management* (pp. 432-446). (Lecture Notes in Computer Science). https://doi.org/10.1007/978-3-319-49004-5_28

Published in:
20th International Conference on Knowledge Engineering and Knowledge Management

Citing this paper
Please note that where the full-text provided on Manchester Research Explorer is the Author Accepted Manuscript or Proof version this may differ from the final Published version. If citing, it is advised that you check and use the publisher's definitive version.

General rights
Copyright and moral rights for the publications made accessible in the Research Explorer are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

Takedown policy
If you believe that this document breaches copyright please refer to the University of Manchester's Takedown Procedures [http://man.ac.uk/04Y6Bo] or contact uml.scholarlycommunications@manchester.ac.uk providing relevant details, so we can investigate your claim.

Download date:30. Jun. 2022

Making Entailment Set Changes Explicit Improves the Understanding of Consequences of Ontology Authoring Actions

Nicolas Matentzoglu, Markel Vigo, Caroline Jay, and Robert Stevens

The University of Manchester
Oxford Road, Manchester, M13 9PL, UK
{nicolas.matentzoglu,markel.vigo,caroline.jay,robert.stevens}@manchester.ac.uk

Abstract. The consequences of adding or removing axioms are difficult to apprehend for ontology authors using the Web Ontology Language (OWL). Consequences of modelling actions range from unintended inferences to outright defects such as incoherency or even inconsistency. One of the central ontology authoring activities is verifying that a particular modelling step has had the intended consequences, often with the help of reasoners. For users of Protégé, this involves, for example, exploring the inferred class hierarchy.
We explore the hypothesis that making changes to key entailment sets explicit improves verification compared to the standard static hierarchy/frame-based approach. We implement our approach as a Protégé plugin and conduct an exploratory study to isolate the authoring actions for which users benefit from our approach. In a second controlled study we address our hypothesis and find that, for a set of key authoring problems, making entailment set changes explicit improves the understanding of consequences both in terms of correctness and speed, and is rated as the preferred way to track changes compared to a static hierarchy/frame-based view.

Keywords: ontology engineering, ontology authoring, reasoning

1 Introduction

Ontologies are explicit conceptualisations of a domain, and are widely applied in biology, health-care and the public domain. Ontologies are typically represented in a formal representation language such as the Web Ontology Language (OWL), the Open Biomedical Ontologies format (OBO) or the RDF Schema language (RDFS). The central advantage of using such formalisms is their well-defined semantics. Generic reasoning systems can be used to access knowledge in the ontology that is only implied, i.e. not explicitly stated, allowing richer answers to queries, the identification of inconsistent knowledge and improved management of large terminologies through definition-oriented development. There is strong, if mainly anecdotal, evidence that building ontologies using OWL is difficult and error-prone.

Attempts were made to quantify this difficulty [4], but an accurate model of the cognitive complexity of various ontology authoring tasks such as exploration or modelling has yet to be defined. The cognitive complexity of OWL can lead to axioms that do not reflect the intentions of the author. Furthermore, the rich and often complicated semantics of OWL can result in unintended inferences, which are often not made explicit by the authoring tool, and even if they are, are rarely communicated to the author clearly. An interview study recently revealed that many ontology experts *frequently* run the reasoner, sometimes *after every modification*, to detect errors such as unsatisfiable classes and to prevent the spread of errors [12]. Participants also felt that the change evaluation phase, i.e. the phase that determines whether a modelling action had the intended consequences, is not well supported by state of the art development tools. Some ontology authors use DL queries, generated on the fly, to do 'spot checks', others work with competency questions that are crafted upfront to automatically verify the correctness of a change. As the conceptual model of an ontology is, however, not always known upfront, competency question based approaches, perhaps best compared with unit tests in software engineering, unfold their utility later in the engineering process and their coverage of the ontology depends on the user's diligence. Consequently we need the user interface to remove this complexity from the ontologies, support the evaluation of ontologies and to either prevent or detect errors.

In this work, we are concerned with improving the evaluation of modelling actions. In the context of this work, we call the task of evaluating that a particular modelling action has had the desired effect "verification". Verification is a key sub-process of ontology authoring that involves conducting a set of tests, for example to make sure that a definition of a class works as intended and that no unsatisfiable classes were introduced [2]. When developing ontologies with the popular Protégé ontology engineering environment, the verification step is typically realised by invoking the reasoner and exploring the implicit knowledge in the ontology [12], for example by making sure that a particular class has the expected position in the inferred class hierarchy or a freshly introduced property domain restriction results in the expected individual type inferences. We call this approach *static hierarchy/frame-based* (SHFB), where "static" refers to the fact that the inferred hierarchy only reflects a state, without any indication how this state relates to the latest modelling action. We explore the hypothesis that making changes to a number of key entailment sets explicit improves verification compared to the static hierarchy/frame-based approach. Our contributions are as follows:

- We developed the Inference Inspector, a novel Protégé plugin that makes changes to key entailment sets as consequences of modelling actions explicit.
- We conducted an exploratory study to evaluate our Inference Inspector prototype. We find that our approach is better suited for tasks that involve change, such as changing definitions or adding restrictions, and less well

suited for tasks that involve the introduction of new entities compared to SHFB and is well received by users for a number of typical modelling tasks, in particular changing class definitions.
– We conducted a laboratory experiment that confirms our hypothesis. We find that making entailment set changes explicit improves the understanding of consequences both in terms of correctness and speed, and is rated as the preferred way to track changes compared to SHFB.

2 Background and Related Work

Ontology authoring is the creation and maintenance of ontology artefacts represented in a formal knowledge representation language such as OWL, OBO or RDFS. We view an ontology $\mathcal{O}$ as a set of axioms, and α with $\alpha \in \mathcal{O}$ being an axiom in $\mathcal{O}$. The signature of $\mathcal{O}$ is the set of entities across all axioms in $\mathcal{O}$. Typical ontology authoring activities include, but are not limited to, the creation of axioms or annotations. For a detailed discussion of ontology authoring activities see [14]. Research on ontology authoring has experienced a resurgence in recent years [12, 14, 15]. One reason for this might be the increased utilisation of change-logs for ontology development. WebProtégé for example produces change-logs during ontology authoring, which can form the basis of rich and informative analyses on ontology authoring activities [15].

While ontology authoring is increasingly performed in a programmatic fashion, a large number of ontologies have been built using ontology authoring environments such as Protégé [5] and WebProtégé [11]. Moreover, even if ontologies are created in a programmatic fashion, they are often checked for defects in a visual authoring environment. The work presented here is the continuation of a series of investigations into the processes of ontology authoring [14, 13, 12]. The aim of the series is to improve our understanding of ontology authoring processes, in particular to identify typical authoring styles and workflows to guide tool developers to improve their support of those workflows. We identified typical problems during ontology authoring, in particular that Protégé does not cater for all the needs of current authors [12, 13]. Ever more sophisticated ontology modelling patterns make the verification of modelling actions difficult. Unintended consequences such as the introduction of unsatisfiable classes, broken definitions (that result in wrong classifications) or wrong inferences on the data level (ABox) are often difficult to spot, which was one of the core incentives for this work. A specially modified version of Protégé that collects interaction events silently during ontology authoring [14], Protégé4US, enabled us to study ontology authoring workflows and derive a number of well-founded design suggestions for authoring tools [14]. One of these was *making the changes to the inferred hierarchy explicit*— another major incentive for developing the Inference Inspector.

The existing tool support for ontology authoring activities is still largely inadequate [12]. An example of an early study that established the necessity of presenting explanations for entailments and reporting errors adequately in the context of knowledge representation (KR) systems was McGuiness et al. [8]. Ontology authoring tools continued to receive poor usability ratings [3, 6, 12] throughout the last 20 years. Examples of unmet demands from users include the ability to compare different versions of the ontology [3] and inadequate debugging support [12]. In particular, making the consequences of modelling actions explicit beyond simply identifying that a defect exists has received little attention. The main effort in this direction came from Denaux et al. [2]. The authors developed a system that provides interactive semantic feedback directly after a change to the ontology. They suggest 6 categories of semantic feedback from the ontology engineering environment, given a single axiom α being added to the ontology: α was already asserted, α was not asserted, but could be inferred, α causes the ontology to be inconsistent, α is novel and the addition results in new implications, α is novel and the addition does not result in new implications, and α causes a concept in the ontology to become unsatisfiable. While Denaux et al. inspired us to produce a better feedback mechanism, their work differs in two fundamental aspects to the research presented here: (1) in Denaux et al. only additions are modelled, i.e., the case that an engineer adds an axiom to the ontology, while we also cover removals, and (2) changes are comprised of a single axiom, while we decided to model sets of additions and removals. Only providing feedback when the reasoner is run returns the responsibility for asking for feedback to the engineer, keeps the interface responsive (reasoning is not required after every step), but also comes with a caveat: given a set of changes, it may not be anymore possible to attribute a particular inference (either lost or gained) to a particular change, thereby putting the burden of identifying the erroneous change back to the engineer. We believe, however, that the gain in responsiveness is worth this caveat, and we can cover some of these shortcomings using justifications, as explained in the next section. The authors evaluate their approach using a task based setting similar to the exploratory study we present later, and find that the feedback was generally considered helpful. However, no formal evaluation was conducted to find out whether the feedback actually led to more accurate modelling. The tool is available online (`https://sourceforge.net/projects/entendre/`).

3 Inference Inspector: Making the Consequences of Modelling Actions Explicit

We present the Inference Inspector, a Protégé plugin for making the consequences of modelling actions in an ontology explicit. The Inference Inspector is implemented as a plugin for Protégé 5 (5.0.0 at the time of writing). We consider ontologies to be represented in OWL 2 DL, unless otherwise stated. A *modelling action* is defined as a non-empty set of changes $\mathcal{CH}$. A *change* can either be a removal of an axiom α, denoted R_α, or an addition, denoted A_α. For example, given the addition of an axiom α_1: SubClassOf(A, B) and the removal of another axiom α_2: SubClassOf(A, C), the modelling action is defined as $\mathcal{CH} : \{A_{\alpha_1}, R_{\alpha_2}\}$. Axiom *modifications* are always treated as an addition of the revised axiom and a removal. In the previous example, the user might have decided that A should not be subsumed by B, but by C instead, changing the existing SubClassOf(A, B) to SubClassOf(A, C). The Inference Inspector makes changes to a predefined set of key entailment sets explicit. A change to an entailment set is defined as follows. Given an $\mathcal{O}$, a previous version of the ontology $\mathcal{O}^{'}$ and a finite entailment set $\mathcal{E}$, the difference between the respective entailment set of $\mathcal{O}$ and $\mathcal{O}^{'}$, $\mathcal{E}_{\mathcal{O}} \setminus \mathcal{E}_{\mathcal{O}'}$, is called the set of *added inferences w.r.t.* $\mathcal{E}$, and the difference between the entailment set of $\mathcal{O}^{'}$ and $\mathcal{O}$, $\mathcal{E}_{\mathcal{O}'} \setminus \mathcal{E}_{\mathcal{O}}$ is called the set of *removed inferences w.r.t.* $\mathcal{E}$. Given a language $\mathcal{L}$ and an OWL 2 ontology $\mathcal{O}$, the $\mathcal{L}$-entailment set of $\mathcal{O}$, written $\mathcal{E}(\mathcal{O}, \mathcal{L})$, is the set of all axioms in $\mathcal{L}$ that are entailed by $\mathcal{O}$ (entailment set).

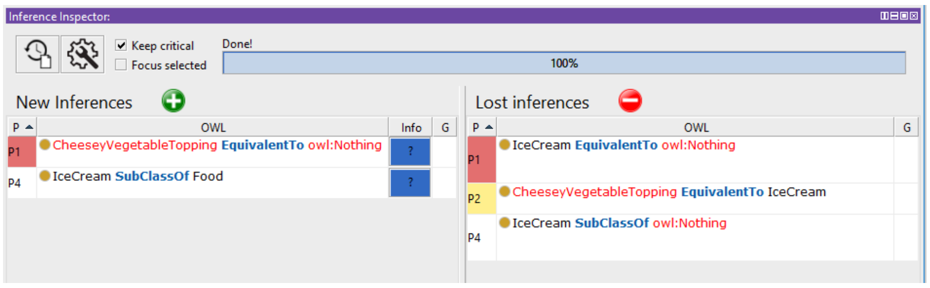
There are a number of factors that inform the selection of appropriate entailment sets for presentation [10]. Our approach was fairly practical: the entailments shown should help the user to verify their modelling actions and not be too costly to compute. In order to help the user verifying their modelling actions, the entailment set should be indicative of erroneous or correct modelling and be easily understandable by a typical user. In order to be *indicative of erroneous modelling*, the presented entailments should be verifiable against modelling intentions. Since we cannot directly access the user's modelling intentions, we make a number of simplifying assumptions. Firstly, we consider unsatisfiable classes or ontology inconsistency as bugs, which any user aims to avoid. Secondly, we assume that the majority of users have a mental model of the hierarchical structures of their ontology, including a model of class disjointness, and intend to keep the ontology consistent with that mental model. In other words, we assume that the ontology author knows, for a concept they are modelling, where in the hierarchy it should be situated and which individuals should be members of it, as well as whether it is disjoint from another concept in the domain. Therefore, we primarily serve the modelling intentions of avoiding bugs while producing hierarchies consistent with that mental model. We acknowledge that this assumption is not universally true, as it is, for example, unlikely that any one author of the gene ontology [1], for example, knows all subsumptions between all the concepts it covers. Furthermore, there are other relevant axes that are not covered by our approach, such as partonomy or any class level patterns that are based on object properties, such as existential restrictions. We do, however, believe that the subsumption relation is of central importance in a majority of cases, which is also confirmed by our finding that users look at the class hierarchy 45% of the time spent editing ontologies with Protégé [14]. Furthermore, the presented entailments must be *easily understandable* by a user, i.e. we do not want to replace a cognitively demanding search, such as a lookup in a large hierarchy, by a cognitively demanding parse, such as an axiom involving deeply nested class expressions. Therefore, our approach only considers entailments that involve named entities, and avoid those that involve complex class expressions such as existential restrictions. Lastly, determining the entailment sets should not be too computationally expensive. Current implementations of subsumption and instantiation perform well in practice [9], despite the high worst case complexity. Computing the full set of disjoint classes, on the other hand, can be more computationally intensive in practice, because reasoners do not implement efficient algorithms for this task. We therefore allow the user to determine whether computing the disjointness relation is worth their while.

We consider the following (groups of) entailment sets: (1) atomic subsumptions between classes, and object and data properties; (2) equivalences between classes, and object and data properties; (3) object property characteristics; (4) disjointness between class names; (5) class assertions and; (6) object property assertions. While (1), (2), (4) and (5) directly correspond to the considerations above, (3) and (6) do not. We include object property assertions in our solution in order to provide a mechanism that allows the user to check whether sub-object property chains (and the object property hierarchy) work as intended. The reason for including object property characteristics was that our extensive experience teaching novice and advanced OWL users has shown that the inheritance or non-inheritance of object property characteristics up or down the object property hierarchy is extremely difficult to understand. For example, making an object property functional makes all its children's properties functional, while the same is not true for transitivity. Although the entailment sets considered here are finite, they are potentially large. To further reduce the amount of information shown to the user, for the three atomic subsumption entailment sets, we consider the transitive reduct, i.e. we query the reasoner only for direct subsumptions.

After the selection of appropriate entailment sets, the second question that is relevant when presenting entailments to users according to Parvizi *et al.* [10] is how they should be ordered. The Inference Inspector implements a configurable system for inference prioritisation. We allow the user to assign a priority to an item from a list of pre-defined inference patterns. Currently, we have implemented five priority levels and 11 inference patterns. The priority levels range from critically important (ontology defects such as unsatisfiability) to unimportant (e.g. asserted axioms).

By default, the Inference Inspector orders the presented consequences by priority. Sometimes, ordering the potentially large number of entailments presented to the user is not enough. In particular, inferences on the ABox (individual) level can be extremely numerous. We therefore employ a grouping strategy for object property assertion axioms and class assertions [10]. By default, we group all axioms of the type ClassAssertion(a,X), where X is a particular class name in the ontology, and ObjectPropertyAssertion(a, b, R), where R is a particular object property name in the ontology. For very large ontologies the list of inferences can be further narrowed down by restricting it to particular entities in the ontology. Justifications for entailments can be computed on demand.

After each reasoner run, a snapshot of the current ontology is created including its inferences. By default users are presented with the consequences of their most recent modelling action (Figure 1). We define the scope of a single modelling action as the set of all changes that were applied to the ontology between the latest and the previous run of the reasoner. For example, after running the reasoner, the ontology author might add three axioms and remove one. When


Inference Inspector:

☑ Keep critical
☐ Focus selected

Done!
100%

New Inferences

P	OWL	Info	G
P1	CheeseyVegetableTopping **EquivalentTo** owl:Nothing	?	
P4	IceCream **SubClassOf** Food	?	

Lost inferences

P	OWL	G
P1	IceCream **EquivalentTo** owl:Nothing	
P2	CheeseyVegetableTopping **EquivalentTo** IceCream	
P4	IceCream **SubClassOf** owl:Nothing	

Fig. 1: A snapshot of the inference inspector after the removal of an axiom that made IceCream unsatisfiable. Left: we can see the new position of IceCream in the class hierarchy. Right: we can see respective lost inferences. P1 (Critical), P2 (important) and P4 (not important) are priority levels (P).

the user runs the reasoner again, they will be presented with the entailments added and lost since the previous run of the reasoner. From an implementation perspective, this is realised by computing the set difference in accordance with the definitions given in the beginning of this section. The Inference Inspector allows the user to compare the current state of the ontology to any snapshot created previously. The first snapshot is always the empty ontology: this means that comparing the current version of the ontology with the empty ontology will always show all inferences (short of those that are explicitly hidden by the user). By default, inferences of critical importance (P1 in Figure 1) are always shown, no matter which snapshot forms the basis for comparison, but this feature can be switched off, if only the latest changes are of interest. Lastly, the user may restrict the inferences presented, by either: (1) showing only inferences related to the currently selected entity in Protégé or (2) showing only inferences involving entities manually selected in a special entity selector panel. An important caveat of the Inference Inspector implementation is that it relies on the correctness of the reasoner. Reasoners are not always correct [7], and may not support the inference of all the entailment sets considered by the Inference Inspector.

4 Materials and Methods

We conducted two studies to evaluate our approach. The first was an exploratory study performed in the context of an advanced OWL modelling tutorial (E1), intended to evaluate our prototype and isolate modelling actions where authors may benefit from the Inference Inspector. A second, controlled laboratory experiment (E2) validated the hypothesis that making changes to key entailment sets explicit improves modelling performance.

4.1 E1: Prototype Evaluation

Goals The main goals of this study were to evaluate the Inference Inspector prototype and determine those modelling actions where our approach is likely to add value over existing solutions. The evaluation was designed to be broad and involved rating the Inference Inspector for perceived usefulness and responsiveness, as well as providing feedback on the user interface. From the results we extracted modelling actions where our approach may help, and used this information to design tasks for the second experiment (Section 4.2).

Participants 15 intermediate users of Protégé were recruited in the context of a two day advanced OWL tutorial (see `http://ow.ly/pK8P300x9wq`). An Amazon Voucher (£10) was given to those that were willing take part. Most participants had successfully completed a beginner level OWL tutorial or had an equivalent experience with OWL. Out of the 15 (9 female) participants, there were 3 students, 5 PhD students, 3 research fellows, 1 data researcher, 1 assistant director for information management, 1 clinician, 1 information architect and 1 bioinformatician. The mean self-reported expertise level (Likert-scale, 1,Novice - 5,Expert) was 2.47 (standard deviation 0.99) for Protégé and and 2.53 (standard deviation 1.06) for OWL.

Experimental Setup Participants were asked to perform 10 typical ontology authoring tasks in the context of an ontology about pizza (726 axioms,$\mathcal{SHOIN}$) using 5 pre-defined tabs in Protégé. A task typically involved an action, such as adding a definition and running the reasoner, an act of exploration, such as inspecting the changes that occurred, answering one or more control questions about this inspection and finally rating all five tabs for the suitability of performing the task and/or the inspection. The five pre-designed tabs were: a simple list of inferred axioms ("Inferences" view in Protégé), the Protégé "Classes" tab, the Inference Inspector, the Protégé "Individuals" tab and the DL Query tab of the DL Query plugin for Protégé. For navigation purposes, all views showed an asserted class hierarchy on the left hand side, which participants were instructed to ignore when evaluating the suitability of the views for each task. The ten tasks were: (1) understanding the topic of the ontology, (2) identifying unsatisfiable classes, (3) repairing unsatisfiable classes, (4) verifying the repair of an unsatisfiable class, (5-6) verifying the definition of a new class, (7) changing the definition of an existing class, (8) verifying the loosening of a restriction by removing a disjointness axiom, (9) verifying the change of an object property assertion and (10) verifying the addition of a role chain. To avoid participant bias we presented the Inference Inspector simply as a third-party plugin, rather than our own work. Participants were not formally introduced to the Inference Inspector prior to the experiment, but some of the basic functionality was covered as part of the preceding OWL Tutorial. The study took around 50 minutes.

4.2 E2: Making Changes to Key Entailment Sets Explicit Improves Verification Performance

Goals The goal of the second study was to verify the following hypothesis: *Making gained and lost entailments explicit improves the user's understanding of consequences of authoring actions compared to a hierarchy/frame-based view.*

Improvement of user understanding was tested through a series of exploration questions and is evaluated using the following metrics:

- Correctness of understanding: (#true positives + #true negatives)/#options
- Speed of understanding: time to completion / correctness
- Ease of understanding: #mouse-click / correctness, scroll time / correctness
- User suitability rating: Likert-scale (0,unusable-4,perfectly usable)
- Breakdown of answer options, i.e. the number of user-selected answers that are correct (true positives), wrong (false positives) and the number of answers not selected by the user that are correct (true negatives) and wrong (false negatives).

The speed score accounts for the correctness of the answer: the more incorrect it is, the higher the penalty on the score.

Participants 19 (5 female) participants were recruited via word-of-mouth and email-advertisement. The background of the participants ranged from MSc students and intermediate experience to academics and non-academic professionals with a high level of OWL expertise between 22 and 57 years of age (mean 33.28). Out of the 19 participants, there were 5 students, 5 PhD students, 5 academics and 4 non-academics. 4 reported to be involved with ontologies as ontology engineers, 8 as ontology researchers and 6 as ontology tool developers (1 other). The mean self-reported expertise level (Likert-scale, 1,Novice-5,Expert) was 3.53 (standard deviation 0.61) for Protégé and and 3.68 (standard deviation 0.75) for OWL.

Experimental Setup The controlled study was conducted in a designated usability lab. All participants used the same machine with a 24 inch monitor with Protégé 5.0.0 installed. Protégé was pre-configured with the Inference Inspector and the specially developed Protégé Survey Tool [1] (PST) to administer the survey. Tasks were designed for the following problem areas: tightening conceptualisation (adding restrictions, verifying consequences), loosening conceptualisation (removing restrictions, fixing unsatisfiable classes) and changing conceptualisation (changing class definitions). In order to mitigate the impact of varying user expertise levels, all tasks were designed in pairs, i.e. two very similar tasks were designed with one being tested using the Inference Inspector and the other one being tested using the Classes or the Individuals tab. No task required access to the properties tabs. Tabs were assigned to tasks (1 task of each pair to the Inference Inspector) using a Latin square, and then randomly sampled. The survey contained a total of 14 verification tasks. The TBox focused tasks were presented in a scenario involving an ontology about pizza (604 axioms,$\mathcal{SHOIN}$), and the ABox focused tasks were presented in a scenario involving an ontology about family history (89 axioms,$\mathcal{SHIF}$). The participant was asked to answer 2-3 exploration questions for every task, most of which were of the sort "Did the class hierarchy change?" or "Which are the new subclasses of X?" In order to increase participant focus, all questions were auto-submitted after 60 seconds. The questions were designed to be answerable comfortably within that timeout by a reasonably experienced user of Protégé. Answers submitted that way were counted as if they were submitted in the regular way. The study had a maximum duration of 50 minutes.

[1] `https://github.com/matentzn/protegesurvey`

5 Results and Discussion

5.1 E1: Prototype Evaluation

Figure 2 shows which views users considered their preferred option to tackle an ontology authoring problem. The participant was allowed to select a single view that was considered the most adequate for addressing a problem. For identifying unsatisfiable classes, at least 5 participants rated the Inference Inspector as the preferred view, compared to 8 who preferred the Classes tab. The Inference Inspector presents unsatisfiable classes clearly to the user, but so does Protégé. The result suggests that at the very least, for this important task, the Inference Inspector is in fact usable. The same argument goes for the repair of unsatisfiable classes, which both views support by allowing the user to delete axioms occurring in a justification. With respect to adding and changing definitions or restrictions, we expected the Inference Inspector to outperform, because it makes the changes explicit and not subject to a potentially complicated search in the class hierarchy. When, however, users were asked to explore the consequences of defining a new named class, they preferred the class hierarchy. This may be because changes with respect to the defined class are made explicit simply by its position in the hierarchy: all sub- and super-classes are new. Only a third of participants preferred the Inference Inspector for this task, possibly because a visual representation of the class hierarchy is easier to understand than a list of axioms. The Inference Inspector did add value, however, when it came to understanding the consequences of a change in the definition of an existing class (i.e. the EquivalentClasses axiom). Seven users preferred the Inference Inspector to six who preferred the class hierarchy, possibly because it is harder to detect a change in the position of a class, than it is to see the introduction of a new one. The first prototype of the Inference Inspector did poorly on problems involving individuals, perhaps because ABox inferences were not ordered or grouped in any way, which was improved for the subsequent study.

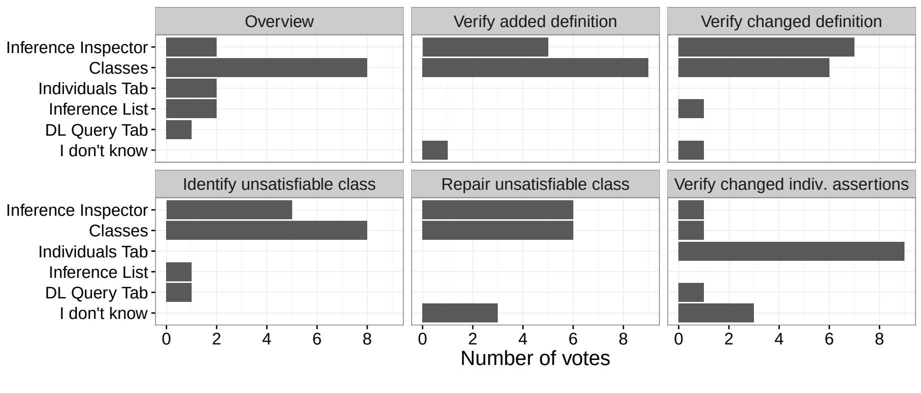


Fig. 2: Preferred view for addressing problem, one vote per participant.

Figure 3 shows the distribution of scores, on a 5 point Likert scale, the Inference Inspector received for key usability criteria. Ease of use was the weakest point of the Inference Inspector (mean rating of 2.93). While this can be partially attributed to a lack of familiarity with the tool ("it seems useful but I don't understand it"), participants also found the plugin a "little busy, layout-wise", "overwhelming to use at first", and that there "are maybe too much information/options in the same view" (see next paragraph on free-form feedback). As a consequence of this feedback, we reduced the options shown to the user and adopted features such as the axiom renderer from Protégé to ensure a more familiar look and feel before conducting our second user study. Reliability had a mean rating of 3.47, with at least 4 participants giving it a low rating of 2. The problem with reliability may also have resulted from a lack of familiarity—users may have not been sure what to expect, and therefore been confused by the feedback. Response time was generally rated good (4.27, despite the Inference Inspector being several seconds slower than the reasoner in Protégé. It remains to be seen how well the Inference Inspector scales. Users expressed interest in using the Inference Inspector for their own work (3.87) and would recommend it to others (3.87).

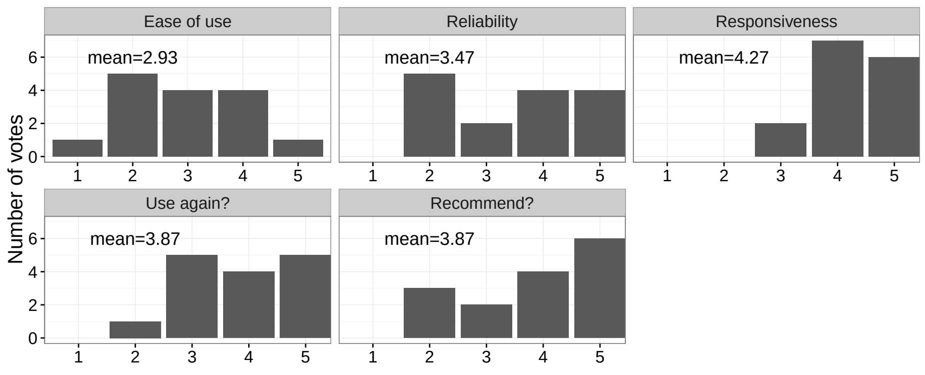


Fig. 3: Ratings of Inference Inspector (1: unusable, 5: perfectly usable)

Study participants were also asked to provide some free-form feedback. They recognised the importance of prioritising and reducing the information shown (independent of whether the Inference Inspector succeeded at this): searching the class hierarchy might be "very tedious in a [..] large ontology [..]." and "some means to filter and sort the output" is required, for example by "prioritizing inferences related to unsatisfiability". Moreover, participants recognised the importance of immediate feedback: "it'd be helpful to have more indication that the repair was successful. As of now, we're looking for the absence of error warnings. If protege could compare the states and flash a message saying 'fixed' [..], it'd be better." This is exactly the sort of feedback we are trying to provide. As the participants were not told about the authors affiliation with the tool, there was little risk of experimenter bias. Unsurprisingly the feedback on the Inference Inspector was mixed. The Protégé Classes tab was rated as favourite for most (TBox related) tasks: "It's the golden standard for all Protégé views", "I felt this view was the most helpful for editing and [..] information." Participants, however, also recognised the potential impact of familiarity bias: "Maybe I like it because I am familiar with it". It is possible the familiarity bias had a significant impact on results, especially since participants were not formally introduced to the Inference Inspector.

5.2 E2: Making Changes to Key Entailment Sets Explicit Improves Verification Performance

We tested the potential for improvement of verification performance provided by the Inference Inspector with respect to (1) a particular range of tasks and (2) the consequences of a change. While we generalise our results for tasks of the kind described in Section 4.2, we do not say anything about other kinds of modelling tasks, such as actions that involve datatypes. Secondly, we are interested in understanding the consequences of a change. This means we do not measure the performance of exploration tasks such as "What are the superclasses of *A*" or "Which one of the following are not sub-classes of *A*", but instead "Which are the new sub-classes of *A*" and "Which subclasses were lost to *A*". The difference is subtle, but important. We use a Wilcoxon signed-rank test to determine whether the difference between the Inference Inspector and the respective Protégé views is statistically significant (at a significance level of p=0.05) for a particular metric.

Exploration tasks were more likely to be performed correctly with the Inference Inspector (80%), than with the Protégé views (65%, p=0.009), see Table 1. This provides evidence that our hypothesis, for the specified set of tasks, holds. At a closer look (Figure 4) we can see that while the problems solved with the Inference Inspector are clustered at the high end of correctness, the ones solved with Protégé are more evenly spread. The same can be observed for the user ratings. We acknowledge, however, that subjective ratings are potentially unreliable due to experimenter bias. Tasks were also performed faster with the Inference Inspector; in the case of exploration tasks, the difference was more than 4.5 sec (mean). However, this difference is not statistically significant (p=0.095). If answer correctness is taken into account (speed), the difference is even greater (and significant, p=0.017). Figure 4 shows how the distribution of task performance time (speed) with the Inference Inspector is shifted to the left. The distribution of the task duration, in particular the high density of short-duration tasks, can be explained by the immediacy with which questions such as "Which classes are unsatisfiable?" or "Did the class hierarchy change?" can be answered with the Inference Inspector. Figure 4 shows how the level to which users needed to scroll is distributed. While there are more tasks that require very little scrolling when using the Inference Inspector, there are also some tasks that require a lot—for some problems, the Inference Inspector shows the results immediately, for others, searching is required. The difference between the Inference Inspector and Protégé however is not statistically significant (p=0.155). The primary form of navigation in the Protégé hierarchy is expanding the nodes rather than scrolling, which manifests itself typically as mouse-clicks. There were considerably less clicks involved in arriving at a correct answer (p=0.015). Looking at false positives and false negatives gives a more fine grained picture of correctness. False positives are wrong observations, i.e. question options that were false but selected by the user. False negative answers are missed answers, that suggest that the view gave the participant a somewhat incomplete picture of the consequences. At first glance, it is surprising that the exploration tasks resulted in more false positive explorations than false negative ones (the converse is true for the Inference Inspector). However, this can be explained by a significant number of binary (Yes/No) questions that were frequently answered wrongly in the case of the Protégé views, for example "Did the class hierarchy change?", which is not always obvious when using the "Classes" tab.

	mean		*median*		*sd*		*p*
Metric	II	P	II	P	II	P	
Correctness	0.80	0.65	1.00	0.75	0.32	0.34	*0.009*
Rating	3.51	1.93	4	2	0.82	1.41	*0.003*
Duration	30.25	34.82	23.96	32.13	20.90	23.11	0.095
Speed	43.04	59.31	28.16	43.31	47.04	58.24	*0.017*
Ease (mouse-click)	6.75	11.09	3.00	8.00	10.59	12.03	*0.015*
Ease (scroll-amount)	2.83	5.74	0.00	0.00	13.72	19.38	0.155
False negatives	0.04	0.10	0	0	0.21	0.29	*0.002*
False positives	0.05	0.25	0	0	0.22	0.43	*0.002*
True negatives	0.79	0.56	1	1	0.41	0.50	*0.002*
True positives	0.95	0.73	1	1	0.22	0.44	0.141

Table 1: Mean, median and standard deviations for key metrics. II is the Inference Inspector, P is Protégé (depending on the task, either individual or Classes tab). P-values larger than 0.05 indicate that differences are not statistically significant.

6 Conclusions

Ontologies can be complex systems of axioms, and a modelling action may have consequences throughout the whole system. Being able to apprehend these consequences should be useful in ontology authoring. We presented the Inference Inspector–a tool that shows the consequences of modelling actions–and explored the hypothesis that making changes to key entailment sets explicit improves understanding of the consequences of such actions.

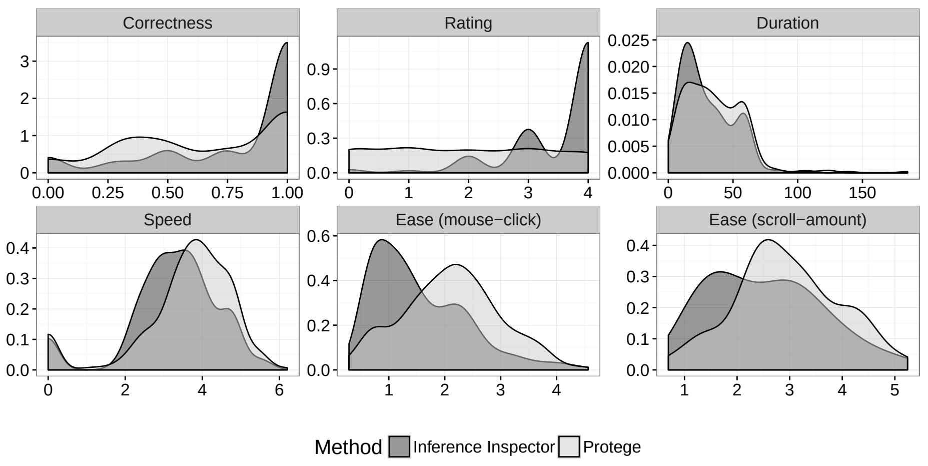


Fig. 4: Kernel density plots for 6 key metrics (x: metric, y: density). Lower three log-rescaled.

We find that making entailment set changes explicit improves understanding of the consequences of a range of key modelling actions. We provide evidence that the standard static view of an ontology does not adequately support people in understanding the consequences of modelling actions, and should be addressed by current ontology authoring environments. Making the consequences of changes explicit as changes to entailment sets is by no means the only, or necessarily best, way to approach this issue. For example, we can easily imagine solutions that highlight changes to the class hierarchy directly. We hope, however, that our work shows that making changes explicit is a key feature missing from ontology authoring environments based on the static hierarchy/frame-based paradigm and that the Inference Inspector will help ontology authors to verify their modelling choices more easily, thereby improving the ontology authoring experience. The plugin is actively maintained and available at `https://github.com/matentzn/inference-inspector`. We welcome bug reports and feature requests to be submitted through GitHub's issue tracking system. A demonstration video, along with links to the source code, the tutorial and the results of both studies, can be found at `http://ow.ly/pK8P300x9wq`.

Acknowledgments: This research has been funded by the EPSRC project WhatIf: Answering "What if..." questions for Ontology Authoring, EPSRC reference EP/J014176/1.

References

1. T. G. O. Consortium. Gene Ontology Annotations and Resources. *Nucleic Acids Research*, 41(D1):D530–D535, 2013.
2. R. Denaux, D. Thakker, V. Dimitrova, and A. G. Cohn. Interactive Semantic Feedback for Intuitive Ontology Authoring. In *Formal Ontology in Information*

Systems - Proceedings of the Seventh International Conference, FOIS 2012, Gray, Austra, July 24-27, 2012, pages 160–173, 2012.

3. M. Dzbor, E. Motta, C. Buil, J. M. Gomez, O. Görlitz, and H. Lewen. Developing Ontologies in OWL: an Observational Study. In *Proceedings of the OWLED*06 Workshop on OWL: Experiences and Directions, Athens, Georgia, USA, November 10-11, 2006*, 2006.
4. M. Horridge, S. Bail, B. Parsia, and U. Sattler. The Cognitive Complexity of OWL Justifications. In *The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I*, pages 241–256, 2011.
5. H. Knublauch, R. W. Fergerson, N. F. Noy, and M. A. Musen. The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. In *SEMWEB*, 2004.
6. P. Lambrix, M. Habbouche, and M. Pérez. Evaluation of ontology development tools for bioinformatics. *Bioinformatics*, 19(12):1564–1571, 2003.
7. M. Lee, N. Matentzoglu, B. Parsia, and U. Sattler. A Multi-reasoner, Justification-Based Approach to Reasoner Correctness. In *The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part II*, pages 393–408, 2015.
8. D. L. McGuinness and P. F. Patel-Schneider. Usability Issues in Knowledge Representation Systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, IAAI 98, July 26-30, 1998, Madison, Wisconsin, USA.*, pages 608–614, 1998.
9. B. Parsia, N. Matentzoglu, R. S. Gonçalves, B. Glimm, and A. Steigmiller. The OWL Reasoner Evaluation (ORE) 2015 Competition Report. In *Proceedings of the 11th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS-2015), Bethlehem, Pennsylvania, USA, October 11, 2015.*, 2015.
10. A. Parvizi, C. Mellish, K. v. Deemter, Y. Ren, and J. Z. Pan. Selecting Ontology Entailments for Presentation to Users. In *KEOD 2014 - Proceedings of the International Conference on Knowledge Engineering and Ontology Development, Rome, Italy, 21-24 October, 2014*, pages 382–387, 2014.
11. T. Tudorache, C. I. Nyulas, N. F. Noy, and M. A. Musen. WebProtégé: A collaborative ontology editor and knowledge acquisition tool for the Web. *Semantic Web*, 4:89–99, 2013.
12. M. Vigo, S. Bail, C. Jay, and R. D. Stevens. Overcoming the pitfalls of ontology authoring: Strategies and implications for tool design. *Int. J. Hum.-Comput. Stud.*, 72(12):835–845, 2014.
13. M. Vigo, C. Jay, and R. Stevens. Design insights for the next wave ontology authoring tools. In *CHI Conference on Human Factors in Computing Systems, CHI'14, Toronto, ON, Canada - April 26 - May 01, 2014*, pages 1555–1558, 2014.
14. M. Vigo, C. Jay, and R. Stevens. Constructing Conceptual Knowledge Artefacts: Activity Patterns in the Ontology Authoring Process. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, CHI 2015, Seoul, Republic of Korea, April 18-23, 2015*, pages 3385–3394, 2015.
15. H. Wang, T. Tudorache, D. Dou, N. F. Noy, and M. A. Musen. Analysis of User Editing Patterns in Ontology Development Projects. In *OTM 2013 Conferences*, pages 470–487. Springer Berlin Heidelberg, Sept. 2013.