A Protégé Plug-In for Test-Driven Ontology Development

Konstantin Schekotihin, Patrick Rodler, Wolfgang Schmid
University of Klagenfurt
Universitätsstr. 65-67
Klagenfurt, Austria
Email: firtstname.lastname@aau.at

Matthew Horridge, Tania Tudorache
Stanford University
1265 Welch Rd
Stanford, California, USA
Email: lastname@stanford.edu

***Abstract*—Ontology development is a hard and often error-prone process, which requires ontology authors to correctly express their domain knowledge in a formal language. One way to ensure the quality of the resulting ontology is to use test cases, similarly to the best practices in software development. For ontology development, test cases can be specified as statements describing expected and/or unwanted logical consequences of an ontology. However, verifying the test cases and identifying the ontology parts that cause their violation is a complex task, which requires appropriate tool support.**

In this demo, we present OntoDebug—a plug-in for the Protégé editor—that supports test-driven ontology development. OntoDebug can automatically verify whether the ontology satisfies all defined test cases. If any test case is violated, the plug-in assists the user in debugging and repairing the ontology in an interactive way. The plug-in asks a series of questions about the ontology to pin point the faulty axioms. Once a fault is repaired, all answers that the author provided in the interactive debugging session, may be converted into test cases, thus preserving the additional knowledge, which can be used in future testing of the ontology.

I. Test-Driven Ontology Development

Semantic applications, especially in biomedicine, depend on high-quality ontologies. Prior studies in psychology found that it is generally hard for humans to formulate correct logical descriptions [1]. More recent studies indicate that this observation holds also in the context of ontology development, and that available ontology editing tools lack appropriate support for quality assurance [2], [3].

Test-driven ontology development is a paradigm that has its roots in the best-practices of software development. The main idea is to enable ontology authors to specify test cases to ensure the correctness of the ontology with respect to the intended meaning. Each test case is represented as an ontology axiom, and describes some required or unwanted logical consequence of the ontology.

Rector et al. [4] identified several unintended consequences in a study of SNOMED CT, for example, "feet are a part of pelvis", or "diabetes is a disease of the abdomen". Such unintended consequences, rooted in faulty modeling, can be prevented by using a test-driven development approach. For the first example ("feet are a part of pelvis"), a developer may add test cases ensuring that different body regions are disjoint and, thus, find a modeling error: "The dorsalis pedis artery is a part of the abdomen and pelvis." The verification of test cases as well as the localization and repair of axioms causing their violation is very difficult without appropriate tool support.

This work was partially supported by the Carinthian Science Fund (contract KWF-3520/26767/38701).

II. The OntoDebug Protégé Plug-In

To address this tooling gap, we developed the OntoDebug Protégé plug-in.[1] It supports the test-driven development of ontologies, and can be installed from the standard plug-ins repository directly in the editor. The user interface of the plug-in is shown in Fig. 1. The functionality of OntoDebug can be summarized as follows:

a) ***Test-driven Development****:* Ontology authors define test cases in the *Original Test Cases* window (Fig. 1, view **5**). The window has two sections: (1) *Entailed Test Cases*, in which users define expected inferred axioms of the intended ontology; and (2) *Non Entailed Test Cases*, in which users specify axioms that must never be inferred. For the specification of test cases it can be helpful to browse the ontology, which can be done in the *Possibly Faulty Axioms* window (Fig. 1, view **3**).

By default, all ontology axioms are viewed as potential sources of fault. However, the ontology author can move an axiom from the *Possibly Faulty Axioms* to the *Correct Axioms* (Fig. 1, view **6**), if the axiom is assumed or known to be correct, and the plug-in should never consider it as faulty.

The *Start/Restart* button triggers the verification of all test cases. If any test case is violated, the plug-in starts a debugging session automatically.

b) ***Debugging of Ontologies****:* OntoDebug implements several ontology debugging algorithms [5], [6], [7], [8], which enable efficient localization of faulty axioms responsible for test case failures. The result of running the debugging algorithm is shown in the *Possible Ontology Repairs* window (Fig. 1, view **4**). An ontology repair from the displayed repair list is a set of axioms that should be changed in order to make all test cases hold. The debugging algorithms use the DL reasoner extensively—for example, to check the consistency of an ontology, or to compute the list of inferred axioms. As the performance of reasoners may vary for different ontologies,

[1] See http://isbi.aau.at/ontodebug for the source code and documentation.

we implemented reasoner-independent debugging techniques that can use any reasoner available in Protégé, such as Pellet [9] or Hermit [10].

c) ***Interactive Debugging****:* The defined test cases are often insufficient to localize the faulty axioms in the ontology. In such a case, the debugger might return several alternative repairs. OntoDebug will help the ontology author find the optimal repair by asking the user a series of questions in an interactive debugging session [11], [12]. The questions are generated automatically and ask the user whether some axioms are logical consequences of the intended ontology or not. For example, in Fig. 1 view **1**, the plug-in asks the ontology author if "KoalaWithPhD is a Koala", and if "KoalaWithPhD is a Person". The user responds affirmatively to the first question, and negatively to the second. The plug-in will then add the user's answers to the list of *Acquired Test Cases* (Fig. 1, view **2**), and it will refine the set of repairs by removing those that violate the newly acquired answers. The interactive process continues until the developer finds a satisfactory repair.

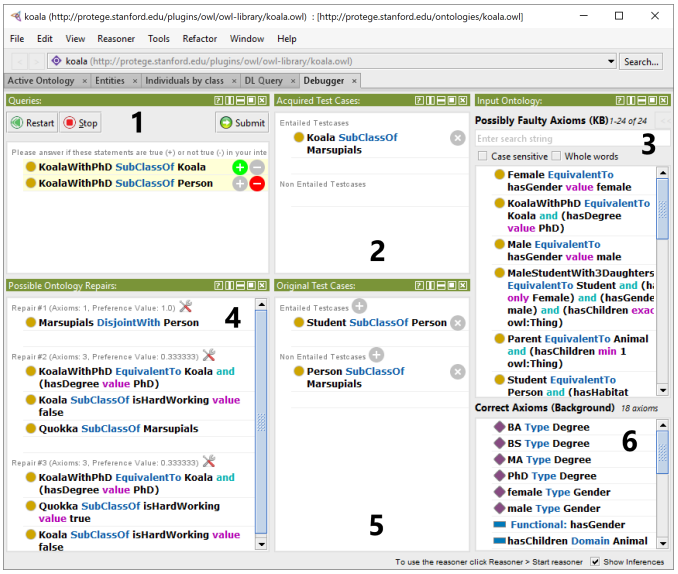

Fig. 1. Interactive ontology debugging session in OntoDebug

d) ***Ontology Repair****:* After identifying the right repair (i.e., the faulty axioms), the ontology author can do a dry run of various modifications of these axioms on a copy of the ontology. OntoDebug automatically tests the altered axiom(s) against the specified test cases, and reports *if*, *which* and *why* problems persist (Fig. 2). Once all test cases hold and the ontology author is satisfied with the performed amendments, the changes can be applied to the original ontology.

e) ***Versatile Parametrization and Customization****:* The performance of testing and diagnosis algorithms depends on multiple factors, such as the reasoning complexity of the developed ontology, the number of faulty axioms, and so on. OntoDebug allows the ontology author to control the working of the testing and debugging algorithms by customizing a detailed set of parameters in a Preferences window. If the proper parameters are selected, the performance improvements can be significant.

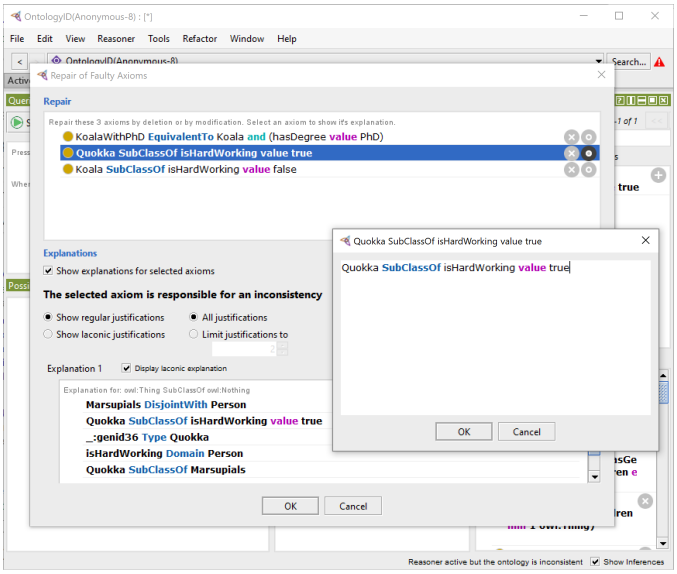

Fig. 2. Repair interface of the OntoDebug plug-in

III. Conclusions

In this demo, we will showcase the OntoDebug plug-in with several use cases. We will demonstrate the different capabilities of the tool, and walk the audience through a complete debug and repair session.

References

[1] P. N. Johnson-Laird, "Deductive reasoning," *Annu Rev Psychol*, vol. 50, pp. 109–135, 1999.

[2] M. Vigo, S. Bail, C. Jay, and R. Stevens, "Overcoming the pitfalls of ontology authoring: Strategies and implications for tool design," *Int. J. Hum.-Comput. Stud.*, vol. 72, no. 12, pp. 835–845, 2014.

[3] A. Rector, N. Drummond, M. Horridge, J. Rogers, H. Knublauch, R. Stevens, H. Wang, and C. Wroe, "OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors & Common Patterns," in *EKAW*, pp. 63–81, 2004.

[4] A. Rector, S. Brandt, and T. Schneider, "Getting the foot out of the pelvis: modeling problems affecting use of SNOMED CT hierarchies in practical applications," *JAMIA*, vol. 18, no. 4, pp. 432–440, 2011.

[5] M. Horridge, B. Parsia, and U. Sattler, "Laconic and Precise Justifications in OWL," in *ISWC*, pp. 323–338, 2008.

[6] K. M. Shchekotykhin, G. Friedrich, P. Rodler, and P. Fleiss, "Sequential diagnosis of high cardinality faults in knowledge-bases by direct diagnosis generation," in *ECAI*, pp. 813–818, 2014.

[7] K. M. Shchekotykhin, D. Jannach, and T. Schmitz, "Mergexplain: Fast computation of multiple conflicts for diagnosis," in *IJCAI*, pp. 3221–3228, 2015.

[8] P. Rodler, W. Schmid, and K. Schekotihin, "Inexpensive cost-optimized measurement proposal for sequential model-based diagnosis," in *DX Workshop*, pp. 200–218, 2018.

[9] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical OWL-DL reasoner," *J. Web Sem.*, vol. 5, pp. 51–53, 2007.

[10] B. Motik, R. Shearer, and I. Horrocks, "Hypertableau Reasoning for Description Logics," *JAIR*, vol. 36, pp. 165–228, 2009.

[11] K. M. Shchekotykhin, G. Friedrich, P. Fleiss, and P. Rodler, "Interactive ontology debugging: Two query strategies for efficient fault localization," *J. Web Sem.*, vol. 12, pp. 88–103, 2012.

[12] P. Rodler, "Interactive Debugging of Knowledge Bases," Ph.D. thesis, Alpen-Adria Universität Klagenfurt, 2015. [Online]. Available: http://arxiv.org/pdf/1605.05950v1.pdf