The OWL API: A Java API for Working with OWL 2 Ontologies

Matthew Horridge[1], Sean Bechhofer[1]

The University of Manchester, UK

Abstract. This paper presents the OWL API a high level Application Programming Interface (API) for working with OWL 2 ontologies. The API is closely aligned with the OWL 2 structural specification. It supports parsing and rendering in the syntaxes defined in the W3C specification, namely, the Functional Syntax, RDF/XML, OWL/XML and the Manchester OWL Syntax. Finally, the reference implementation of the API, which is written in Java, includes validators for the various OWL 2 profiles - OWL 2 QL, OWL 2 EL and OWL 2 RL.

1 Introduction

Aside from the fact the OWL is a W3C Recommendation [15], one of the probable reasons for its success and relatively broad adoption in academia and industry is that there have been a wide range of OWL tools available. These tools have supported the creation and editing of OWL ontologies, reasoning over ontologies, and using ontologies in applications. Generally speaking, all of these tools require some kind of underlying API that allow ontologies to be loaded, manipulated and queried. In the case of ontology editors, such as Protégé-4 [7], the NeOn Toolkit [6], Swoop [10], TopBraid Composer[1] and OWLSight[2], it is usually a necessity to have some mechanism to load ontologies in various concrete formats, along with clean mechanisms to manipulate and modify ontologies. With regard to reasoning, client applications usually require general purpose reasoner interfaces so that they can easily swap in and out different reasoner implementations. On the other side of the coin, reasoner developers can benefit from being able to provide a well known interface to their reasoners.

The latest version of the OWL API has been designed to meet the needs of people developing OWL based applications, OWL editors and OWL reasoners. It is a high level API that is closely aligned with the OWL 2 specification. It includes first class change support, general purpose reasoner interfaces, validators for the various OWL 2 profiles, and support for parsing and serialising ontologies in a variety of syntaxes. The API also has a very flexible design that allows third parties to provide alternative implementations for all major components.

The purpose of this paper is to discuss the major components and functionality of the API and to present it as the API of choice for anyone working with OWL 2 ontologies.

[1] `http://www.topquadrant.com/products/TB_Composer.html`
[2] `http://pellet.owldl.com/ontology-browser`

2 OWL API Design

The original version of the OWL API provided a suite of interfaces along with a reference implementation that facilitated the use of OWL in a wide variety of applications. The underlying design rationale for that API was discussed in detail in [2]. Although the latest version of the OWL API still follows much of the approach outlined in [2], it has undergone a number of significant design changes in order to ensure alignment with the OWL 2 specification. In particular, one of the main changes has been a shift from the underlying the frame oriented model that was provided by the OWL Abstract Syntax to an axiom oriented model upon which the OWL 2 standard is based.

At its core, the OWL API consists of a set of interfaces for inspecting, manipulating and reasoning with OWL ontologies. The OWL API supports loading and saving ontologies is a variety of syntaxes. However, none of the model interfaces in the API reflect, or are biased to any particular concrete syntax or model. For example, unlike other APIs such as Jena [3], or the Protégé 3.X API, the representation of class expressions and axioms is not at the level of RDF triples. Indeed, the design of the OWL API is directly based on the OWL 2 Structural Specification [13]. This means that an ontology is simply viewed as a set of axioms and annotations as depicted in Figure 1. The names and hierarchies of interfaces for entities, class expressions and axioms in the OWL API correspond closely to the structural specification. In fact, there is almost a one-to-one translation between the OWL 2 Structural Specification and core OWL API model interfaces, meaning that it is easy to relate the high level OWL 2 specification directly to the design of the API.

2.1 Ontology Management

Besides the model interfaces for representing entities, class expressions and axioms, it is necessary to have certain machinery that allow applications to manage ontologies. Figure 1 shows a high level overview of this picture. The `OWLOntology` interface provides a point for efficiently accessing the axioms contained in an ontology. For example, axioms can be accessed by type and by signature to name a few. Different implementations of the `OWLOntology` interface can provide different storage mechanisms for ontologies. While the reference implementation of the API provides an efficient main-memory storage solution, it is possible to provide alternative implementations. This is discussed further in Section 2.4.

The `OWLOntologyManager` provides a central point for creating, loading, changing and saving ontologies, which are instances of the `OWLOntology` interface. Each ontology is created or loaded by an ontology manager. Each instance of an ontology is unique to a particular manager, and all changes to an ontology are applied via its manager.

This centralised management design allows client applications to have one access point to ontologies, to provide redirection mechanisms and other customisations for loading ontologies, and allows client applications to monitor all changes that are applied to any loaded ontologies. The manager also hides much of the complexity associated with choosing the appropriate parsers and renderers for loading and saving ontologies and therefore obviates clients from worrying about these issues.

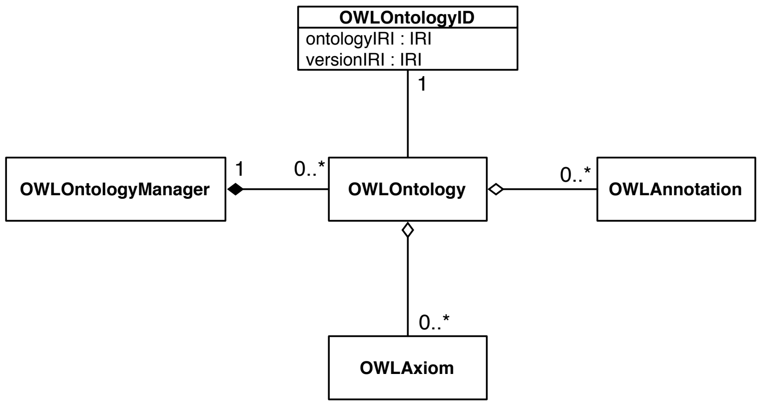


Fig. 1. A UML diagram showing the management of ontologies in the OWL API. The `OWLOntology` interface provides accessors for efficiently obtaining the axioms contained within an ontology. The method of storing axioms is provided by different implementations of the `OWLOntology` interface. The design of the API makes it possible to mix and match implementations, for example, an in-memory ontology could be used together with one that is stored in a database and one that is stored in some kind of triple store. The OWL API reference implementation provides an efficient in-memory storage solution.

2.2 Ontology Change

Changes to ontologies are applied using change objects that implement the `OWLOntologyChange` interface. Besides adding and removing axioms, changes to ontologies include setting the ID of an ontology, adding and removing annotations, and adding and removing imports.

In essence, change support is implemented using the *Command* design pattern [4], which makes it easy to record and serialise ontology changes, implement undo and redo functionality, and support transactional behaviour. This makes the API ideal for use in editors and other systems that need the ability to log changes.

All ontology changes are applied through an ontology manager. This means that it is possible to apply a list of ontology changes, which make multiple changes to multiple ontologies, as a single unit. This works rather well for applications such as ontology editors, where an edit operation such as entity name change (entity IRI change) can involve the addition and removal of multiple axioms from multiple ontologies—it is possible to group the changes together to form one “editor operation” and apply these changes at one time.

2.3 Parsing and Rendering OWL Ontologies

A major benefit of aligning the OWL API with the OWL 2 structural specification is that there is no commitment to a particular concrete syntax. While there is only one syntax that an OWL implementation *must* support, namely RDF/XML, there are several other syntaxes that exist which are optimised for different purposes. For example, Turtle syntax provides a slightly more readable RDF serialisation. Similarly, the Manchester OWL Syntax provides a human readable serialisation for OWL ontologies. OWL/XML [12] is a newly designed syntax that allows ontologies to be stored in "plain" XML that, unlike RDF/XML, can be used directly by XML tools and technologies such as XPath.

The OWL API therefore includes out of the box support for reading and writing ontologies in several syntaxes, including RDF/XML, Turtle, OWL/XML, OWL Functional Syntax, The Manchester OWL Syntax, KRSS Syntax and the OBO flat file format. Due to the underlying design of the API, it is possible for the imports closure of an ontology to contain ontologies that were parsed from ontology documents written in different syntaxes.

The reference implementation of the OWL API uses a registry of parsers and renderers, which makes it easy to add support for custom syntaxes. The appropriate parser is automatically selected at runtime when an ontology is loaded. By default, ontologies are saved back into the format from which they were parsed, but it is possible to override this in order to perform syntax conversion tasks and "save as" operations in editors for example.

2.4 Data Structure Storage

The reference implementation provides data structures for efficient in-memory representations of ontologies. For many purposes this is sufficient. For example, the latest versions of the National Cancer Institute (NCI) ontology can comfortably fit into around 400MB of memory (100MB of memory without annotations). However, the API has been designed so that it is possible to provide other storage mechanism for ontologies. For example, there are use cases for storing ontologies in relational databases or triple stores. The OWL API has been designed with these use cases in mind, and it is possible to "mix and match" storage implementations, so that an ontology imports closure could contain in-memory representations of ontologies, ontologies persisted in secondary storage in the form of a custom database, and ontologies stored in triple stores.

While the API does not include these alternative storage mechanisms out of the box, third parties have developed such solutions. An exemplar solution, called OWLDB, has been developed by Henss et al [9]. Their solution stores ontologies in a relational database, using a "native" mapping of OWL constructs (as opposed to a mapping to triples) to a custom database schema.

2.5 Reasoner Interfaces

A key part of working with OWL ontologies is reasoning. Reasoners are used to check the consistency of ontologies, check to see whether the signature of an ontology contains unsatisfiable classes, compute class and property hierarchies, and check to see if axioms are entailed by an ontology. The OWL API has various interfaces to support interacting with OWL reasoners. The main interface is the `OWLReasoner` interface which provides methods to perform the aforementioned tasks.

The reasoner interfaces have been designed so as to make it easier for reasoners to expose functionality that provides incremental reasoning support. The API allows a reasoner to be set up so that it listens for ontology changes and either immediately processes the changes or queues them in a buffer which can later be processed. This design caters for the scenario where a reasoner is used within an ontology editor and, at some point, must respond to edits of the ontology, or situations where a reasoner should respond to ontology changes as they arrive.

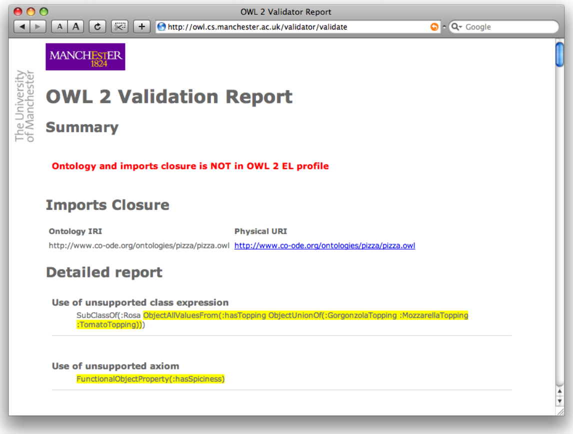
At the time of this writing, the CEL [1], FaCT++ [17], HermiT [14], Pellet [16], and Racer Pro [5] reasoners provide OWL API wrappers. This means that they are also available as reasoner plugins to Protégé-4.

3 Profile Validation

The OWL 2 specification provides various OWL profiles that correspond to syntactic subsets of the OWL 2 language. The profiles are defined in the OWL 2 profiles document, namely OWL 2 EL, OWL 2 QL and OWL 2 RL. Each profile is designed to trade some expressive power for efficiency of reasoning. For example, the OWL 2 EL profile trades expressivity for the benefit of polynomial time subsumption testing. Similarly, reasoning for the OWL 2 RL profile can be implemented using a rule engine.

For people that use these profiles, and tools that support them, it can be necessary to determine whether or not an ontology falls into one of the profiles or not. The OWL API contains an API to deal with ontology profiles. Various profile related interfaces are available that provide functionality to ask whether an ontology is within a profile. When doing this a profile report is generated that specifies whether an ontology and its imports closure fall into a given profile, and if not details *why* this is the case. The profile API allows complete programmatic access by client software, with fine-grained objects that represent specific reasons for profile violations.

A Web based application that performs profile validation on an ontology and its imports closure was written with the OWL API is available at `http://owl.cs.manchester.ac.uk/validator`. A screen shot of a report generated by the validator is shown in Figure 2. Each item in the detailed report can be accessed programmatically, which means that client software can customise report rendering, or offer more advanced functionality such as repair suggestions that would take an ontology back into the desired profile.



OWL 2 Validator Report

http://owl.cs.manchester.ac.uk/validator/validate

OWL 2 Validation Report

Summary

Ontology and imports closure is NOT in OWL 2 EL profile

Imports Closure

Ontology IRI	Physical URI
http://www.co-ode.org/ontologies/pizza/pizza.owl	http://www.co-ode.org/ontologies/pizza/pizza.owl

Detailed report

Use of unsupported class expression

```
SubClassOf(:Rosa ObjectAllValuesFrom(:hasTopping ObjectUnionOf(:GorgonzolaTopping :MozzarellaTopping
:TomatoTopping)))
```

Use of unsupported axiom

```
FunctionalObjectProperty(:hasSpiciness)
```

Fig. 2. A Screenshot of the OWL 2 Profile Validator. The validator is written using the Profiles API which is part of the OWL API

4 Changes between Version 2 and Version 3

The OWL API has undergone several significant changes between version 2 and version 3, which will have an impact on client code. These changes are described in the following sections.

4.1 Ontology Identity

In previous versions of the API, ontologies were identified by IRI (URI). However, since OWL 2, it is not necessary to "name" ontologies with an IRI. Additionally, if an ontology does have an IRI then it may also have a version IRI. The OWL API therefore provides an abstraction of ontology identity using an "identifier object", namely `OWLOntologyID`. Ontology IDs are immutable, but the ID of an ontology may be set at any time via the ontology's manager.

4.2 Axiom Annotations

The previous version of the OWL API supported axiom annotations, however the mechanism for creating and managing axiom annotations has changed significantly. Originally, axiom annotations were "stand-off" objects. That is, an axiom annotation had an independent existence from the axiom it annotated. In line with this, the structural identity of an axiom was not affected by any annotations on the axiom. In the final OWL 2 specification, axiom annotations are *embedded* into the axiom itself and affect the structural identity of the axiom. For example, `SubClassOf(Annotation(rdfs:comment ‘‘Added on 09/09/2009’’) A B)` is not structurally equal to the axiom `SubClassOf(A B)`. The OWL API was modified to remove the disparity with the OWL 2 specification. While seemingly minor, this modification does have an impact on the way that annotations are added and removed from axioms. Suppose a user of a tool such as Protégé-4 wants to annotate, `SubClassOf(A B)` with `Annotation(rdfs:comment ''Added on 09/09/2009'')`. While the user may think of the annotation process as simply adding the annotation to the axiom, and this was the model used in the previous version of the OWL API, it is now necessary to remove `SubClassOf(A B)` from the ontology and create a new axiom `SubClassOf(Annotation(rdfs:comment ''Added on 09/09/2009'') A B)` which is added to the ontology.

4.3 Interface Naming

Interfaces for representing entities, class expressions, and axioms have been named in accordance with the OWL 2 Structural Specification. Unfortunately this has meant that some interface name changes were necessary. For example, in previous version of the API, the high level interface for representing class expression was called `OWLDescription`, but this has been renamed to `OWLClassExpression`. There are obvious benefits of having the API interface names aligned with the names used in the OWL 2 structural specification. In particular, the extensive, and high quality, suite of OWL 2 specification documents serve as secondary OWL API documentation. After reading the OWL 2 specification documents, people should be able to easily identify the correspondence between the API and the specification.

While these name changes have introduced a backwards incompatibility with previous versions of the API, it was felt that the benefits of directly following the OWL 2 specification, and aligning the names of API interfaces with the names used in the specification outweighed the disadvantages. Additionally, the changes are such that a simple find and replace should be possible, and it is likely that a converter will be provided to do the bulk of updating existing OWL API 2.X client code to version 3.0.0 of the API.

5 Examples of Use

As an illustration of the kinds of applications and services that can be developed using the OWL API some examples are discussed below. The examples have been split into use of the API for editors and browsers, and use in OWL oriented services.

5.1 Use in Editors and Ontology Browsers

The OWL API is used in the following editors and browsers:

– Protégé-4— An open source OWL ontology editor that was initially designed and developed at the University of Manchester. Protégé-4 uses the OWL API to underpin all ontology management tasks, from loading and saving ontologies, to manipulating ontologies during editing, to interacting and offering a choice of OWL reasoners. Virtually all of the functionality provided by the OWL API is utilised by Protégé-4.

– The NeOn Toolkit— An Eclipse based ontology development environment that was developed as part of the 14.7 million Euro EU funded project. While early versions of the toolkit were written on top of KAON2[3], the project has recently switched over to using the OWL API.
– OWLSight — A web based ontology browser written by Clark & Parsia that uses the Pellet reasoner. The browser is written using the Google Web Toolkit, with the OWL API being used to read and access ontologies.
– Ontology Browser —- An ontology browser that dynamically generates documentation for ontologies and is based on the OWLDoc software. The OWL API is used for loading and accessing ontologies and interfacing with the FaCT++ reasoner.
– OntoTrack —- A browsing and editing tool for OWL ontologies [11] that is developed at Ulm University. The OWL API is used for loading and accessing ontologies for rending into a graph.

5.2 Use in OWL Services

– Syntax Converter — A web based application that converts ontologies written in one OWL syntax to another OWL syntax. Syntax is converted from one format to another format by loading and saving ontologies using the OWL API. (`http://owl.cs.manchester.ac.uk/converter`)
– Ontology Repository — A repository of ontologies used for reasoner and tools testing. The loading and serialising makes use of the OWL API parsers and renderers, while the metrics for each ontology are computed by the OWL API's Metrics API. (`http://owl.cs.manchester.ac.uk/repository`)
– Validator — Uses the various profile validators to determine if an ontology and its imports closure is within a specific profile. The validator returns valdiation reports in a variety of human readable syntaxes. The validator makes use of the Profiles API, which is part of the OWL API. (`http://owl.cs.manchester.ac.uk/validator`)
– Module Extractor — Allows locality based modules to be extracted from ontologies. The extractor makes used of OWL API modularisation code, which currently provides Syntactic Locality Based Modules [8]. (`http://owl.cs.manchester.ac.uk/modularity`)

6 Conclusions

– The latest version of the OWL API supports the OWL 2 specification
– It provides support for parsing and rendering ontologies written in RDF/XML, Turtle, OWL/XML, OWL Functional Syntax, and Manchester OWL Syntax.

[3] `http://kaon2.semanticweb.org`

– The API provides a common interface to various reasoners, including CEL, FaCT++, HermiT, Pellet and Racer Pro.
– OWL 2 Profile validation functionality is included in the distribution. Ontologies can be checked to see if they are OWL 2, OWL 2 DL, or in the OWL 2 EL, OWL 2 QL, or OWL 2 RL profiles.
– The OWL API may be downloaded via `http://owlapi.sourceforge.net`.

7 Acknowledgements

The authors would like to acknowledge the many people who have contributed to the OWL API since the first version was designed and developed by Sean Bechhofer, Raphael Volz and Philip Lord [2] in 2003. In particular (in alphabetical order), Ron Alford[4], Sebastian Brandt[5] Nick Drummond[5], Birte Glimm[6], Luigi Iannone[5], Aditya Kalyanpur[7], Boris Motik[6], Olaf Noppens[8], Pavel Klinov[5], Timothy Redmond[9], Thomas Schneider[5], Evren Sirin[4], Mike Smith[4], past and present members of the Information Management Group (IMG) and the Bio-Health Informatics Group (BHIG) at the University of Manchester, and all of the OWL API users who have posted bug reports and feedback on the OWL API public mailing list.

References

1. F. Baader, C. Lutz, and B. Suntisrivaraporn. CEL—a polynomial-time reasoner for life science ontologies. In U. Furbach and N. Shankar, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR'06)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 287–291. Springer-Verlag, 2006.
2. Sean Bechhofer, Raphael Volz, and Philip Lord. Cooking the semantic web with the OWL API. In Dieter Fensel, Katia Sycara, and John Mylopoulos, editors, *The Semantic Web - ISWC 2003. The Second International Semantic Web Conference, Sanibel Island, Florida, USA*, volume 2870/2003 of *Lecture Notes in Computer Science*, pages 659–675, Sanibel Island, Florida, USA, October 2003. Springer.
3. Jeremy J. Carroll, Ian Dickinson, Chris Dollin, Dave Reynolds, Andy Seaborne, and Kevin Wilkinson. Jena: implementing the semantic web recommendations. In Stuart Feldman, Mike Uretsky, Mark Najork, and Craig Wills, editors, *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 74–83, New York, NY, USA, May 2004. ACM.
4. Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional Computing Series. Addison-Wesley, 21st edition, October 1994.

[4] Clark & Parsia
[5] The University of Manchester
[6] Oxford University
[7] IBM TJ Watson Research Center, New York
[8] Ulm University
[9] Stanford University

5. Volker Haarslev and Ralf Möller. RACER system description. In *International Joint Conference on Automated Reasoning (IJCAR 2001)*, volume 2083 of *Lecture Notes In Computer Science*, pages 701–705, 2001.
6. Peter Haase, Holger Lewen, Rudi Studer, Duc Thanh Tran, Michael Erdmann, Mathieu d'Aquin, and Enrico Motta. The neon ontology engineering toolkit. In Jeff Korn, editor, *WWW 2008 Developers Track*, April 2008.
7. Matthew Horridge, Dmitry Tsarkov, and Timothy Redmond. Supporting early adoption of OWL 1.1 with Protégé-OWL and FaCT++. In Bernardo Cuenca Grau, Pascal Hitzler, Conor Shankey, and Evan Wallace, editors, *OWL: Experiences and Directions (OWLED)*, volume 216 of *CEUR Workshop Proceedings*. CEUR-WS.org, November 2006.
8. Ernesto Jiménez-Ruiz, Bernardo Cuenca Grau, Ulrike Sattler, Thomas Schneider, and Raphael Berlanga Llavori. Safe and economic re-use of ontologies: A logic-based methodology and tool support. In Sean Bechhofer, Manfred Hauswirth, Joerg Hoffmann, and Manolis Koubarakis, editors, *The Semantic Web: Research and Applications, 5th European Semantic Web Conference, ESWC 2008, Tenerife, Canary Islands, Spain*, volume 5021 of *Lecture Notes in Computer Science*, pages 185–199. Springer, June 2008.
9. Joachim Kleb Jörg Henss and Stephan Grimm. A database backend for OWL. In Rinke Hoeksta and Peter F. Patel-Schneider, editors, *OWL: Experiences and Directions (OWLED 2009)*, CEUR Workshop Proceedings. CEUR-WS.org, October 2009.
10. Aditya Kalyanpur, Bijan Parsia, and James Hendler. A tool for working with web ontologies. In *International Journal on Semantic Web and Information Systems*, volume 1, Jan - Mar 2005.
11. Thorsten Liebig and Olaf Noppens. OntoTrack: Combining browsing and editing with reasoning and explaining for OWL Lite ontologies. In Sheila McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *The Semantic Web - ISWC 2004. Third International Semantic Web Conference 2004, Hiroshima, Japan*, volume 3298 of *Lecture Notes in Computer Science*, pages 244–258. Springer, November 2004.
12. Boris Motik, Bijan Parsia, and Peter F. Patel-Schneider. OWL 2 Web Ontology Language XML serialization. W3C Recommendation, W3C – World Wide Web Consortium, October 2009.
13. Boris Motik, Peter F. Patel-Schneider, and Bijan Parsia. OWL 2 Web Ontology Language structural specification and functional style syntax. W3C Recommendation, W3C – World Wide Web Consortium, October 2009.
14. Boris Motik, Rob Shearer, and Ian Horrocks. Optimized reasoning in description logics using hypertableaux. In *Proc. of the 21st Int. Conf. on Automated Deduction (CADE-21)*, volume 4603 of *Lecture Notes in Artificial Intelligence*, pages 67–83. Springer, 2007.
15. Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. OWL Web Ontology Language semantics and abstract syntax. W3C Recommendation, 10 February 2004.
16. Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, 5(2), 2007.
17. Dmitry Tsarkov and Ian Horrocks. FaCT++ description logic reasoner: System description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 292–297. Springer, 2006.