Adapting ontologies to best-practice artifacts using transformation patterns: Method, implementation and use cases

(Original article published in JWS, 2016, Vol. 40, pp 52–64)

Vojtěch Svátek, Marek Dudáš, and Ondřej Zamazal
Semantic Web and Ontological Engineering group (SWOE)
University of Economics, Prague (VSE), Czech Republic

http://kizi.vse.cz/swoe
Acknowledgments

• CSF project on “metamorphosing ontologies” (2010-2012)
• FP7 LOD2 project (2011-2012)
• VSE IGA projects (2014, 2015, 2016)
Warning: slides go slightly beyond the original article content
Broader view of the task: OWL2OWL transformation

• One of the family of tasks aimed having
  • *some* knowledge model on input
  • OWL ontology on output

• The others (cf. Svátek et al., 2016b) are
  • NOWL2OWL – input are, e.g., thesauri, business taxonomies, web directories, DB schemas, ...
  • OBM2OWL – input are *in some sense* more expressive/relaxed models, labeled as „ontological background models“ (OBM), e.g., OntoUML, OntoClean, PURO, ...
Broader view of the task: OWL2OWL transformation

• Why transform OWL again to OWL?

• Overcome limitations of reasoners
  esp. disallowed OWL constructs/constellations

• Better fit to data to be modeled

• Better fit to existing ontologies or other OWL models…
  to be matched to the current ontology
  to be included as (root) part of the current ontology

• Better readability by humans
  esp. introducing best-practice naming conventions
Broader view of the task: OWL2OWL transformation

• Why transform OWL again to OWL?
  • Overcome limitations of reasoners
    • esp. disallowed OWL constructs/constellations
Broader view of the task: OWL2OWL transformation

• Why transform OWL again to OWL?
  • Overcome limitations of reasoners
    • esp. disallowed OWL constructs/constellations
  • Better fit to data to be modeled
Broader view of the task: OWL2OWL transformation

• Why transform OWL again to OWL?
  • Overcome limitations of reasoners
    • esp. disallowed OWL constructs/constellations
  • Better fit to data to be modeled
  • Better fit to existing ontologies or other OWL models...
    • to be matched to the current ontology
    • to be included as (root) part of the current ontology

• Better readability by humans
  • esp. introducing best-practice naming conventions
Broader view of the task: OWL2OWL transformation

• Why transform OWL again to OWL?
  • Overcome limitations of *reasoners*
    • esp. disallowed OWL constructs/constellations
  • Better fit to *data* to be modeled
  • Better fit to *existing ontologies* or other OWL models...
    • to be *matched* to the current ontology
    • to be *included* as (root) part of the current ontology
  • Better readability by humans
    • esp. introducing best-practice *naming conventions*
  • ...

Broader view of the task: OWL2OWL transformation

• Why transform OWL again to OWL?
  • Overcome limitations of reasoners
    • esp. disallowed OWL constructs/constellations
  • Better fit to data to be modeled
  • Better fit to existing ontologies or other OWL models...
    • to be matched to the current ontology
    • to be included as (root) part of the current ontology
  • Better readability by humans
    • esp. introducing best-practice naming conventions
  • ...

JWS article
Broader view of the task: OWL2OWL transformation

• Why transform OWL again to OWL?
  • Overcome limitations of reasoners Šváb-Zamazal et al., 2013
    • esp. disallowed OWL constructs/constellations
  • Better fit to data to be modeled
  • Better fit to existing ontologies or other OWL models...
    • to be matched to the current ontology Šváb-Zamazal et al., 2011
    • to be included as (root) part of the current ontology
  • Better readability by humans Zamazal et al., 2013
    • esp. introducing best-practice naming conventions
  • ...

\[\text{JWS article}\]
Specific task setting

• Ingredient 1: “Legacy” ontology (LO)
  • Developed directly in OWL, or result of an NOWL2OWL process

• Ingredient 2: “Best-practice artifact” (BPA), which can be
  • Best-practice (ontology content) pattern
  • Core ontology
Specific task setting

• Ingredient 1: “Legacy” ontology (LO)
  • Developed directly in OWL, or result of an NOWL2OWL process

• Ingredient 2: “Best-practice artifact” (BPA), which can be
  • Best-practice (ontology content) pattern
  • Core ontology

• (Parts of) the BPA are to become a root-level import of the LO, so as to put in on firmer grounds
Specific task setting

- **Ingredient 1: “Legacy” ontology (LO)**
  - Developed directly in OWL, or result of an NOWL2OWL process

- **Ingredient 2: “Best-practice artifact” (BPA), which can be**
  - Best-practice (ontology content) pattern
  - Core ontology

- (Parts of) the BPA are to become a root-level import of the LO, so as to put in on firmer grounds

- The LO may require a structural transformation so that both models can be merged together
Possible means of OWL2OWL structural transformation

- Manual one-by-one editing in an ontology authoring tool
  - Slow and error-prone
- Ad hoc scripts
  - Low-level solution, scripts might be hard to manage
  - Not every knowledge engineer is a skilful programmer
- SPARQL UPDATE/CONSTRUCT
  - A "middle-way" solution
  - The user cannot interact with the transformation process
  - Support for lexical transformation (of entity names) is limited
  - OWL-specific manipulation languages (OWL-API, OPPL)
    - Still require some program code writing
    - Little support for lexical transformation
    - OPPL no longer maintained
Possible means of OWL2OWL structural transformation

• Manual one-by-one editing in an ontology authoring tool
  • Slow and error-prone
Possible means of OWL2OWL structural transformation

• Manual one-by-one editing in an ontology authoring tool
  • Slow and error-prone
• Ad hoc scripts
  • Low-level solution, scripts might be hard to manage
  • Not every knowledge engineer is a skilful programmer
Possible means of OWL2OWL structural transformation

• Manual one-by-one editing in an ontology authoring tool
  • Slow and error-prone
• Ad hoc scripts
  • Low-level solution, scripts might be hard to manage
  • Not every knowledge engineer is a skilful programmer
• SPARQL UPDATE/CONSTRUCT
  • A „middle-way“ solution
  • The user cannot interact with the transformation process
  • Support for lexical transformation (of entity names) is limited
Possible means of OWL2OWL structural transformation

• Manual one-by-one editing in an ontology authoring tool
  • Slow and error-prone

• Ad hoc scripts
  • Low-level solution, scripts might be hard to manage
  • Not every knowledge engineer is a skilful programmer

• SPARQL UPDATE/CONSTRUCT
  • A „middle-way“ solution
  • The user cannot interact with the transformation process
  • Support for lexical transformation (of entity names) is limited

• OWL-specific manipulation languages (OWL-API, OPPL)
  • Still require some program code writing
  • Little support for lexical transformation
  • OPPL no longer maintained
Possible means of OWL2OWL structural transformation

• Manual one-by-one editing in an ontology authoring tool
  • Slow and error-prone
• Ad hoc scripts
  • Low-level solution, scripts might be hard to manage
  • Not every knowledge engineer is a skilful programmer
• SPARQL UPDATE/CONSTRUCT
  • A „middle-way“ solution
  • The user cannot interact with the transformation process
  • Support for lexical transformation (of entity names) is limited
• OWL-specific manipulation languages (OWL-API, OPPL)
  • Still require some program code writing
  • Little support for lexical transformation
  • OPPL no longer maintained
• Novel approach: using transformation patterns (TPs)
  • PatOMat framework: RESTful services + GUI + pattern editors
  • (Note: First version partially reused OPPL v.2)
Transformation pattern
Transformation pattern

• Structure consisting of
  • Source ontology pattern
  • Target ontology pattern
  • Pattern transformation (PT) specifying the way of transforming a source OP instance to a target OP instance
Transformation pattern

• Structure consisting of
  • Source ontology pattern
  • Target ontology pattern
  • Pattern transformation (PT) specifying the way of transforming a source OP instance to a target OP instance

• Source OP and target OP contain placeholders („variables“) that are instantiated by ontology entities when the source OP is matched against an ontology
Transformation pattern

• Structure consisting of
  • Source ontology pattern
  • Target ontology pattern
  • Pattern transformation (PT) specifying the way of transforming a source OP instance to a target OP instance

• Source OP and target OP contain placeholders („variables“) that are instantiated by ontology entities when the source OP is matched against an ontology

• The *lexical* aspect is handled by
  • Naming detection patterns, as part of the source pattern
  • Naming transformation patterns, as part of the PT
Transformation pattern example
PatOMat transformation workflow

Based on RESTful services
PatOMat transformation workflow

Based on RESTful services
GUI support: GUIPOT Protégé plugin
GUI support: GUIPOT Protégé plugin

Source ontology
GUI support: GUIPOT Protégé plugin

Source ontology

Placeholder instantiations
GUI support: GUIPOT Protégé plugin
Use cases from JWS (2016) paper: adaptation to a best-practice artifact
Use cases from JWS (2016) paper: adaptation to a best-practice artifact

• Adaptation of ontologies modeling the „conference organization“ domain to role-based modeling using the AgentRole content pattern
  • Might allow, e.g., to manage access rights in a review system
Use cases from JWS (2016) paper: adaptation to a best-practice artifact

• Adaptation of ontologies modeling the „conference organization“ domain to role-based modeling using the AgentRole content pattern
  • Might allow, e.g., to manage access rights in a review system

• Adaptation of product ontologies to be structurally compliant with GoodRelations (GR) – a core ontology of e-commerce, now incorporated into schema.org
  • Descriptions of offered products then could be cropped by GR-aware tools
Summary of the testing: „conferences“ use case

- TPs designed and applied by an expert knowledge engineer; for two species of input ontologies
- 16 legacy ontologies (OntoFarm collection)
- Pattern variety
  - Construction of role concepts from either classes or properties
  - Role concepts as either classes or individuals
    - Role names automatically constructed
- Role generation: recall 90% (generation from classes) vs. 83% (generation from properties)
- Omission errors: esp. due to Boolean constructs in domain/range of properties
Summary of the testing: „products“ use case

• TPs designed by an expert knowledge engineer, but applied by novices

• 6 product ontologies
  • From FreeBase, Protégé library, Watson search engine

• 21 transformation patterns for different tasks, some to be applied in a sequence of steps
  • Some specific for FreeBase input
  • Mostly making different properties subproperties of various GR properties and setting their range

• Success ratio of the transformation: between 50-100%, per ontology

• Failure however mostly due to trivial mistakes, such as forgetting to reload the ontology between the steps
When to use TP-based transformation

• Input:
  • medium-sized *ontologies* (e.g., product ontologies)
  • or, ontology *modules* developed by different parties, to be subsequently integrated

• The overall *structure* of the input ontology is
  • not governed by a single pattern
  • but there are some regularities: a couple of simple patterns that can be captured by the transformation pattern

• *Interaction* with the knowledge engineer (esp. filtering the source pattern instances) is needed...

• ... since the targeted use case is *narrower* than can be reliably captured purely by syntactical structures and lexical conventions
When **not** to use TP-based transformation

- Heterogeneous tweaks needed here or there
  - Manual editing is likely better
- Bulk transformation of structurally very simple models (e.g., OWL taxonomy to SKOS taxonomy)
  - SPARQL, or scripting, is likely better
From OWL2OWL to OBM2OWL

• The problem is analogous to machine translation
• The OWL2OWL setting needs a TP for each pair of styles
• The OBM2OWL setting allows one model (OBM) in a more relaxed / expressive language to have the role of interlingua
  • Thus reducing the number of TPs needed
• Our proposed OBM language: PURO (Svatek et al., 2013)
Future directions

• Elaborate the comparison between the alternative approaches to OWL2OWL transformation
  • Incl. testing on users with different background

• Explore novel use cases
  • Such as module adaptation during ontology assembly

• Progress with OBM2OWL transformation
  • Incl. reengineering existing OWL models to their PURO “interlingua” so that they could be transformed to a different encoding style
Selected bibliography


