# Semi-automated Structural Adaptation of Advanced E-Commerce Ontologies

Marek Dudáš[1], Vojtěch Svátek[1], László Török[2],
Ondřej Zamazal[1], Benedicto Rodriguez Castro[2], and Martin Hepp[2]

[1] University of Economics Prague, Nám. W. Churchilla 4, 13067 Praha 3, Czech Rep.
{xdudm12|svatek|ondrej.zamazal}@vse.cz
[2] Univ.der Bundeswehr Munich, W.-Heisenberg-Weg 39, 85579 Neubiberg, Germany
{laszlo.toeroek|benedicto.rodriguez|martin.hepp}@unibw.de

**Abstract.** Most ontologies used in e-commerce are nowadays taxonomies with simple structure and loose semantics. One exception is the OPDM collection of ontologies, which express rich information about product categories and their parameters for a number of domains. Yet, having been created by different designers and with specific bias, such ontologies could still benefit from semi-automatic post-processing. We demonstrate how the versatile *PatOMat* framework for pattern-based ontology transformation can be exploited for suppressing incoherence within the collection and for adapting the ontologies for an unforeseen purpose.

**Key words:** ontology, e-commerce, GoodRelations, transformation, ontology pattern, linked data

## 1 Introduction

The idea that well-designed, structurally rich ontologies would allow to partially automate e-commerce operations has been around for years [1]. Nevertheless, even nowadays, most ontologies exploited in this field are plain taxonomies with imprecise semantics. Proposals for sophisticated modeling remain at the level of academic prototypes, or, at most, are used in closed B2B settings [6].

The *GoodRelations* ontology [3] has then been conceived, as an attempt to balance expressiveness and practical usability, with size comparable to popular linked data vocabularies[1], OWL ontology language[2] expressivity and stress on favorable learning curve thanks to a cookbook with a number of recipes.[3] As 'vertical' extensions to GR, ontologies for specific product/service categories then started to be developed, most recently within the *Ontology-Based Product Data Management* (OPDM) project.[4] The backbone of all ontologies is the taxonomy of product/service types, complemented by taxonomies of product/service features and their enumerated values. This family of ontologies already enjoyed

[1] http://lov.okfn.org/dataset/lov/
[2] http://www.w3.org/TR/owl2-overview/
[3] http://wiki.goodrelations-vocabulary.org/Cookbook
[4] http://www.opdm-project.org/

industrial adoption, such as the car sales ontology used by a major automotive manufacturer.[5]

In this paper we focus on two aspects of such product ontologies for which further enhancement is possible. First, the rapid pace of creation of the ontologies and involvement of multiple designers in parallel occasionally leads to *incoherence* in modeling patterns and naming conventions, both within a single ontology and across a set of them. Second, some of their features are compromises between the best practices for publishing linked data [2] and somewhat different requirements imposed by the e-commerce and web engineering worlds, given they are to be used in direct integration with web-based product catalogs. Therefore they need to be adaptated in order to be used in a 'canonical' linked data setting. In this paper, we label the first issue as *intrinsic incoherence* of an ontology, and the latter as *export-based extrinsic incoherence* (as the ontology is incoherent with the desired target structural style and thus requires adaptation).

As either kind of structural adaptation potentially involves a wide scope of restructuring and renaming operations, it can benefit from application of a user-friendly ontology transformation framework. Such framework has been recently developed under the name of *PatOMat* [9, 11]. In the rest of the paper we first describe the material, i.e., the GoodRelations ontology and the product ontologies based on it (Section 2). Then we present the incoherence problems of both types discovered in the product ontologies (Section 3). Next, the principles of the PatOMat framework and its user interfaces are briefly reviewed (Section 4), and its application on the OPDM ontologies is described (Section 5). Finally, related research is surveyed (Section 6) and the paper is wrapped up (Section 7).

## 2 GoodRelations and Product Ontologies

GoodRelations (further GR) is a generic ontology for e-commerce, which offers conceptual elements to capture facts that are most relevant for exchanging arbitrary goods and services. The core model revolves around the abstraction that an *agent* offers to transfer certain *rights* related to a *product* or *service* [3]. This model is independent of a particular e-commerce domain, since the agent can be any commercial entity making the offer, and rights transferring can range from simple *sale* to *rental* or *leasing*. GR includes generic conceptual elements for products and services and their properties (including prices, delivery or warranty conditions etc.), but no domain-specific product classes or taxonomies.

A premier use case for GR is adding semantic annotation to of e-commerce *web sites*. This allows search engines, novel mobile applications or intelligent browser extensions to analyze and interpret web page semantics at a granularity that would have been otherwise too costly or unreliable. Following this direction, elements of GR have recently been integrated (as officially recognized extension) into *Schema.org*, the unified schema for web page content annotation designed by the leading search engine companies. The two schemas are rather complementary,

[5] `http://www.w3.org/2001/sw/sweo/public/UseCases/Volkswagen/`

as Schema.org has broader coverage of domains (including, e.g., news or recipies) while GR goes much deeper specifically for the e-commerce domain.

Aside the website-level application, there are also domain-specific extensions of GR that can be used within e-commerce *business information systems* as a common data schema that all software services support. Product data available in many systems is often unstructured or incomplete. As sophisticated automated business processes require precise, highly structured data, they are likely to benefit from ontologies capturing data about products from particular domains. *OPDM ontologies*, primarily designed to fulfil this need, extend a subset of GR as in part (b) of Fig. 1: domain-specific *product classes* are subclasses of `gr:ProductOrService`, *product properties* are subproperties of `gr:quantitativeProductOrServiceProperty` or its 'quantitative' or 'datatype' counterparts,[6] and a few *generic properties* such as color, dimension or weight are directly reused from the GR ontology. The ontologies are self-contained, and capture the most frequently occurring properties of each particular product type.

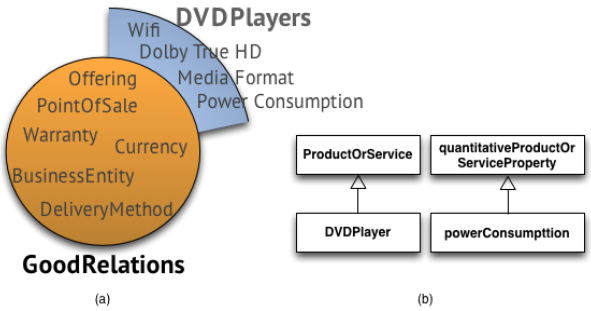


**Fig. 1.** OPDM ontologies as domain-specific extensions of GoodRelations

## 3 Incoherence Problems in OPDM ontologies

### 3.1 Incoherence Types Considered

When an OWL ontology is being developed, there is often more than one option how to model a specific concept or structure, due to high expressiveness of the language. Modeling incoherence may thus arise when such modeling options differ for concepts/structure of similar nature. The fact that OPDM ontologies are all grafted upon the GR ontology somewhat alleviates this problem. Nevertheless, there is still space for incoherence; both at *structural* level, e.g., using a datatype property instead of object property, or at the level of *naming conventions*, such as arbitrarily switching between synonymous lexical constructs.

[6] We omit their full names for typographic reasons – excessive length.

Another way of incoherence classification is according to the situation in which a particular part of an ontology is considered 'incoherent'. Due to the large number of OPDM ontologies and involvement of multiple designers, *intrinsic incoherence* may easily occur, which is a term we suggest for unintentional heterogeneous modeling occurring either within a single ontology or within a collection of ontologies typically presented together (such as the OPDM collection). On the other hand, if the ontologies are to be used outside the original context, it is likely that one will run into what we call *export-based extrinsic incoherence*. Finally, we could also consider *import-based extrinsic incoherence*, which occurs when legacy ontologies have to be adapted to a 'canonical' modeling style (here, the style pre-supposed by GR).[7] In the rest of this section we discuss examples of different types of incoherence in the context of OPDM ontologies.[8]

### 3.2 Intrinsinc Incoherence

This incoherence category is the most critical, since it may impacts any subsequent processing/use of the ontologies. Note that OPDM ontologies are to be filled with large numbers of instances using automated data harvesting methods; this process is difficult if intrinsic incoherence has not been resolved first.

*Intrinsic structural incoherence.* One example of intrinsic structural incoherence is related to modeling the support of various media data types (e.g., GIF, JPEG, AVI etc.) available in an electronic device. There are several ontologies in the OPDM project that cover the described concept (ontologies of computers, cameras, bluray players, portable media players etc.), and as the OPDM ontologies are not modular and are designed to be used independently, the same concept has been designed separately in each ontology. In most of the ontologies there is a class `MediaFormat`, with instances JPEG, GIF, AVI etc., as well as an object property `playbackFormat`, which has the class `MediaType` as its range. In one of the ontologies, however, a different approach is used: there is a boolean data property for each of the media data types. So, for example, the fact that a hypothetical portable media player supports AVI would be expressed as `player playbackFormat AVI` in the former case and as `player AVI true` in the latter. We will refer to this incoherence pattern as to 'boolean vs. instance'.

A similar kind of intrinsic structural incoherence occurred in the case of modeling a property representing port availability in an electronic device. One approach used was to create a boolean data property for each port that might be present on a device, while another was to create an object property for each type of port, which also allows to specify the number of ports of the specified type. The object property has been created as a subproperty of

[7] Pre-cursor work on resolving import-based extrinsic incoherence (though not labeled by this term) at a generic level – with 'canonical' modeling defined by ontology content design patterns – is described in [12].

[8] In all examples, local entities from the individual OPDM ontologies are without prefix, while the GR ontology entities are presented with their usual `gr` prefix.

`gr:quantitativeProductOrService` property with `gr:QuantitativeValue` as its range. The value of the property is then an instance of `gr:QuantitativeValue` with the number of ports specified through the `gr:hasValue` property. The same 'functionality' could have been achieved using an integer-valued data property, but the object property approach is more in accordance with the GR recommendations. Consequently, to model the availability of an HDMI port, according to the first approach we would have a boolean data property called simply `hdmi`, with usage `laptop hdmi true`. According to the other approach we would have an object property `numberOfHdmiPorts` used with an intermediate blank node:

```
laptop numberOfHdmiPorts bnode001.
bnode001 hasValue '2'.
```

We will refer to this incoherence case as to 'boolean vs. integer'.

*Intrinsic naming-level incoherence.* The property representing the port number is also an example of an intrinsic naming-level incoherence. Most of the multiple 'port number property' names in OPDM ontologies follow the pattern `numberOf`*[portType]*`Ports`, but some are named according to the pattern *[portType]*`PortQuantity`. While this kind of incoherence is probably less severe than the structural cases above, its consequences might still be unpleasant, especially if the incoherence arises within the same ontology. We will refer to this incoherence case as to 'infix vs. prefix', as the variable modifier of the name is either in infix or prefix position.

### 3.3 Extrinsic Structural Incoherence

An example of extrinsic structural incoherence comes from considerations of using OPDM ontologies in an 'orthodox' linked data environment. A very relevant opportunity for advanced product ontologies is, for example, their use by an application for *public contracts* management. Efficient matchmaking of calls for tenders and actual tenders (i.e., business offers) could be achieved by specifying the categories and parameters of sought/offered commodities in terms of such ontologies. The *Public Contracts Ontology*[9] designed within the EU LOD2 project, as well as the processing tools that provision RDF data according to this ontology [4], strictly adhere to the linked data principles.

Linked data principles suggest using object properties rather than data properties, since the use of object properties allows for explicitly referring to resources (ontological instances) from external datasets. The OPDM ontologies have not been designed so as to be a part of the linked data cloud. Instead, each OPDM ontology is meant to be used independently, and barriers for their usage by practitioners (unfamiliar with semantic web technologies) is lowered as much as possible, hence most of the properties are datatype properties. This makes them easy to populate with 'instances' in the form of literals; however, in the linked data environment, the benefits of *interlinking* could not be exploited.

[9] `http://code.google.com/p/public-contracts-ontology/`

## 4 PatOMat Framework for Ontology Transformation

The central notion in the *PatOMat* framework[10] is that of *transformation pattern* (TP). A TP contains two *ontology patterns* (the source OP and the target OP) and the description of transformation between them, called *pattern transformation* (PT). The representation of OPs is based on the OWL 2 DL profile, except that *placeholders* are allowed in addition to concrete OWL entities. An OP consists of *entity declarations* (of placeholders and/or concrete entities), *axioms* and *naming detection patterns* (NDP); the last capture the naming aspect of the OP, which is important for its detection. A PT consists of a set of *transformation links* and a set of *naming transformation patterns* (NTP). Transformation links are either *logical equivalence relationships* or *extralogical relationships* (holding between two entities of different type, thus also called 'heterogeneous equivalence'). Naming transformation patterns serve for generating names for target entities. Naming patterns range from *passive naming operations*, such as detection of a head noun for a noun phrase, to *active naming operations*, such as derivation of verb form of a noun. Syntactically, the patterns are expressed according to an XML schema[11] with elements such as `<op1>`, `<op2>` (for the source and target pattern, respectively) and `<pt>` for pattern transformation; deeper in the hierarchy are the `<placeholder>`, `<axiom>` and `<ndp>` elements (for components of OPs) and `<eq>`, `<eqHet>`, `<ntp>` elements for logical equivalence, heterogeneous links and NTPs, within PT). However, the patterns needn't be edited manually, as a graphical editor is available for their authoring.[12]

The framework prototype implementation is available either as a *Java library* or as three *RESTful services*.[13] The Java library is used by the GUIPOT tool[14] (demonstrated in Section 5) and other transformation GUIs. The whole transformation is divided into three steps that correspond to the three core services:

- *OntologyPatternDetection* service takes the TP and an ontology on input, and returns the binding of entity placeholders on output, in XML. The naming detection patterns of the source OP are first processed. As a result of applying the naming aspect, bound placeholders arise that are placed to the FILTER component of a SPARQL[15] query (generated according to axioms in the source OP) before its execution.
- *InstructionGenerator* service takes the binding of placeholders and the TP on input, and returns transformation instructions on output.
- *OntologyTransformation* service takes transformation instructions and the original ontology on input, and returns the transformed ontology on output.

[10] [11] provides details about the initial version of the framework, [9] about the user-oriented tools, and at `http://owl.vse.cz:8080/tutorial/` there is a fully-fledged tutorial for the current version.
[11] `http://nb.vse.cz/~svabo/patomat/tp/tp-schema.xsd`
[12] `http://owl.vse.cz:8080/tpe/`
[13] All accessible via the web interface at `http://owl.vse.cz:8080/`.
[14] `http://owl.vse.cz:8080/GUIPOT/`
[15] `http://www.w3.org/TR/rdf-sparql-query/`

The framework has already been used in multiple use cases, such as:

- Adaptation of the style of an ontology to another one to which it is to be *matched* [11]
- Adaptation of the style of a legacy ontology to a best-practice content pattern being *imported* into it [12]
- *Repair* use cases, including downgrading of an ontology to a less expressive dialect of OWL [10] or entity naming canonicalization [13].

OPDM ontologies represent a challenging mix of problems that have already been touched in the previous use cases. Notably, intrinsic incoherence resolution is related to the matching use case, except that the goal is not to model the same subject proper, but structures with similar semantics. Export-based extrinsic incoherence resolution is related to the repair use cases, since OPDM ontologies are no longer 'best-practice' when put into the linked data context. Finally, import-based extrinsic incoherence resolution would be related to the content pattern import use case (this role being played by the GR ontology here).

## 5 Pattern-Based Transformation of OPDM Ontologies

### 5.1 Selected Transformations in Depth

In this section we present examples of transformation patterns and their application to resolve some of the previously described incoherence cases.

*Transformation for 'boolean vs. instance'* The first incoherence case requires a transformation of boolean data properties to instances of a new class called `MediaFormat`, while also adding a property such as `playbackFormat`, whose range is this class. It can be achieved using the transformation pattern in Fig. 2.[16] The source pattern thereof fits all boolean (as specified in the first axiom) subproperties of `gr:datatypeProductOrServiceProperty` (specified in the second axiom), of which those representing media types have to be selected (currently, manually). The rest of the transformation is performed automatically according to the target ontology pattern and the pattern transformation parts of the transformation pattern, as shown below. The role of the two axioms concerning annotations (labels and comments) is to transfer them to the target transformed ontology. The purpose of the last axiom in the source pattern is to keep the information about the domain of the transformed data property (i.e., some product class) in the placeholder `?pc`. It will be used to set the domain of the newly created object property `playbackFormat`, whose range will be the newly created `MediaFormat` class; its instances arise from the transformed data properties. All the datatype properties `?m` selected in the previous step are transformed into instances `?OP2_m` of class `MediaFormat`, which is created as a new entity. The selected properties `?m` are removed from the ontology and replaced with instances

[16] The | symbols are not part of the code: they only mark elements that are referred to in the explanatory text.

?OP2_m. Axioms describing ?m are also removed except labels and comments (as mentioned above), which are connected to the newly created instances ?OP2_m. The playbackFormat object property (represented by placeholder ?OP2_p) is created, its domain set to ?OP2_pc – the domain of the transformed data property – and its range to ?OP2_C – the newly created MediaClass.

*Transformation for 'boolean vs. integer'.* This kind of incoherence is resolved by transforming boolean data properties selected by the user into object properties that are subproperties of gr:quantitativeProductOrServiceProperty. The pattern is aimed at the described case of 'computer port' properties. Note that the target is the same as in the "infix vs. prefix" naming-level incoherence case from Section 3. The pattern[17] also has the source part structurally similar to the above pattern for 'boolean vs. instance'. A naming transformation is required that adds the numberOf prefix and Ports suffix to the name of the port:

```
<ntp entity="?OP2_m">numberOf+?m+Ports</ntp>
```

*Transformation for 'data vs. object property'.* Data properties[18] are transformed into subproperties of either gr:qualitativeProductOrServiceProperty or its 'quantitative' counterpart, depending on the nature of the property, to keep the ontology GR-compliant. As the transformation pattern framework does not support the if/then/else construct, more than one transformation pattern is needed for this: at least one for qualitative and one for quantitative properties. The qualitative/quantitative character of a property can be recognized from its range (string for qualitative and integer or float for quantitative) – but in the OPDM ontologies, string-valued data properties are sometimes quantitative. To be sure that the transformation is correct, the selection of properties to be transformed has to be checked by the user as in the previous cases, and we need more transformation patterns: one for 'string to qualitative', one for 'string to quantitative', and another two for 'integer to quantitative' and 'float to quantitative'. One more transformation pattern is needed to resolve boolean datatype properties, of which all represent product features in OPDM ontologies. If we do not want to create specific properties as in the 'boolean vs. instance' example, we can create just one hasFeature and transform all boolean data properties to instances of the Feature class created as a subclass of gr:QualitativeValue. Technically, this transformation is analogous to 'boolean vs. instance', except for the name of the new class (which is the general term 'Feature' instead of a specific term such as 'MediaFormat') as well as property (the general 'hasFeature' instead of the specific 'playbackFormat'); also, user interaction is not needed in this case, since all boolean features are transformed without manual selection.

To sum up, the 'data vs. object property' incoherence can be resolved using five transformation patterns. Considering that there are several tens of OPDM ontologies and each has several tens of data properties, the effort invested into designing the transformation patterns is still worth it: it is much faster and easier to use the transformation framework than to change all ontologies by hand.

[17] All patterns are in full extent at http://nb.vse.cz/~svabo/patomat/tp/opdm/.
[18] Unlike the previous cases, all of them in bulk rather than just the selected ones.

```
<op1>
    <entity_declarations>
     <placeholder type="DatatypeProperty">?m</placeholder>
     <placeholder type="Literal">?a1</placeholder>
     <placeholder type="Literal">?a2</placeholder>
     <placeholder type="Class">?pc</placeholder>
     <entity type="Class">&xsd;boolean</entity>
     <entity type="DatatypeProperty">
          &gr;datatypeProductOrServiceProperty</entity>
     <entity type="AnnotationProperty">&rdfs;label</entity>
     <entity type="AnnotationProperty">&rdfs;comment</entity>
    </entity_declarations>
    <axioms>
|    <axiom>DataProperty: ?m Range: boolean</axiom>
|    <axiom>DataProperty: ?m SubPropertyOf:
          datatypeProductOrServiceProperty</axiom>
|    <axiom>DataProperty: ?m Annotations: label ?a1</axiom>
|    <axiom>DataProperty: ?m Annotations: comment ?a2</axiom>
|    <axiom>DataProperty: ?m Domain: ?pc</axiom>
    </axioms>
 </op1>
<op2>
    <entity_declarations>
     <placeholder type="Individual">?OP2_m</placeholder>
     <placeholder type="Class">?OP2_C</placeholder>
     <placeholder type="ObjectProperty">?OP2_p</placeholder>
     <placeholder type="Literal">?OP2_a1</placeholder>
     <placeholder type="Literal">?OP2_a2</placeholder>
     <placeholder type="Class">?OP2_pc</placeholder>
     <entity type="ObjectProperty">
          &gr;qualitativeProductOrServiceProperty</entity>
    </entity_declarations>
    <axioms>
|    <axiom>Individual: ?OP2_m Types: ?OP2_C</axiom>
     <axiom>ObjectProperty: ?OP2_p SubPropertyOf:
          qualitativeProductOrServiceProperty</axiom>
     <axiom>Individual: ?OP2_m Annotations: label ?OP2_a1</axiom>
     <axiom>Individual: ?OP2_m Annotations: comment ?OP2_a2</axiom>
     <axiom>ObjectProperty: ?OP2_p Domain: ?OP2_pc</axiom>
     <axiom>ObjectProperty: ?OP2_p Range: ?OP2_C</axiom>
    </axioms>
 </op2>
 <pt>
    <eqHet op1="?m" op2="?OP2_m"/> <eq op1="?a1" op2="?OP2_a1" />
    <eq op1="?a2" op2="?OP2_a2" /> <eq op1="?pc" op2="?OP2_pc" />
    <ntp entity="?OP2_C">MediaFormat</ntp>
    <ntp entity="?OP2_p">playbackFormat</ntp>
    <ntp entity="?OP2_a1">"+?a1+"</ntp>
    <ntp entity="?OP2_a2">"+?a2+"</ntp>
 </pt>
```

**Fig. 2.** Pattern for transforming (media type) boolean properties to instances

### 5.2 Reusability of the Described Transformation Patterns

In Table 1 we report on possible reusability of the three patterns described above and the (naming) transformation pattern for 'infix vs. prefix' (`Quantity` vs. `NumberOf`) incoherence from Section 3. The table shows the number of OPDM ontologies each transformation pattern can be used for and whether the pattern can be also used for other GR-compliant product ontologies outside of the OPDM project or even for a non-GR-compliant ontology.

**Table 1.** Reusability of the transformation patterns

| **Transformation pattern** | **Number of ontologies** | **Any GR** | **Non-GR** |
|---|---|---|---|
| 'Datatype vs. instance' | 1 (PortableMP) but can be easily adapted | No | Yes |
| 'Boolean vs. integer' | 4 | Yes | No |
| 'Datatype vs. object property' | All (31) | Yes | No |
| 'Infix vs. prefix' | All (31) | Yes | Yes |

### 5.3 Transformation Pattern Application Using GUIPOT

As one of the user-oriented add-ons [9] to the PatOMat framework we developed the Graphical User Interface for Pattern-based Ontology Transformation (GUIPOT), as means for comfortable application of transformation patterns. GUIPOT is a plugin for Protégé. After loading a transformation pattern it displays a list of pattern instances of the source OP detected in the given ontology: see the upper-center of the screen in Fig. 3, for an application of the 'boolean vs. instance' pattern. By selecting one or more instances, the detected entities are highlighted in the hierarchical view of the ontology[19] in the left part of the plugin window. Transformation is then invoked by clicking the 'Apply Transformations' button. The right part of the window shows the ontology after transformation.[20] Entities that were affected by the transformation are highlighted in the hierarchical views.[21] If the user is satisfied with the results of the transformation, s/he can load the transformed ontology into Protégé and continue working with it.

In this example, the class `portablemp:MediaFormat` has been added as a subclass of `gr:QualitativeValue` (upper-right pane) and the object property `portablemp:playbackFormat` has been created (object properties view in the bottom right pane). In the instance list in the bottom middle part of the screen we can see that `portablemp:MediaFormat` has been populated with instances of media formats; and in the data property list next to it can be verified that the properties formerly (see the left part of the screen) representing media formats are no longer present.

[19] They can also be visualized in graph form using the popular OntoGraf plugin.
[20] For brevity, we actually only show the screen as after this step, in in Fig. 3.
[21] Again, they can be also displayed by OntoGraf.

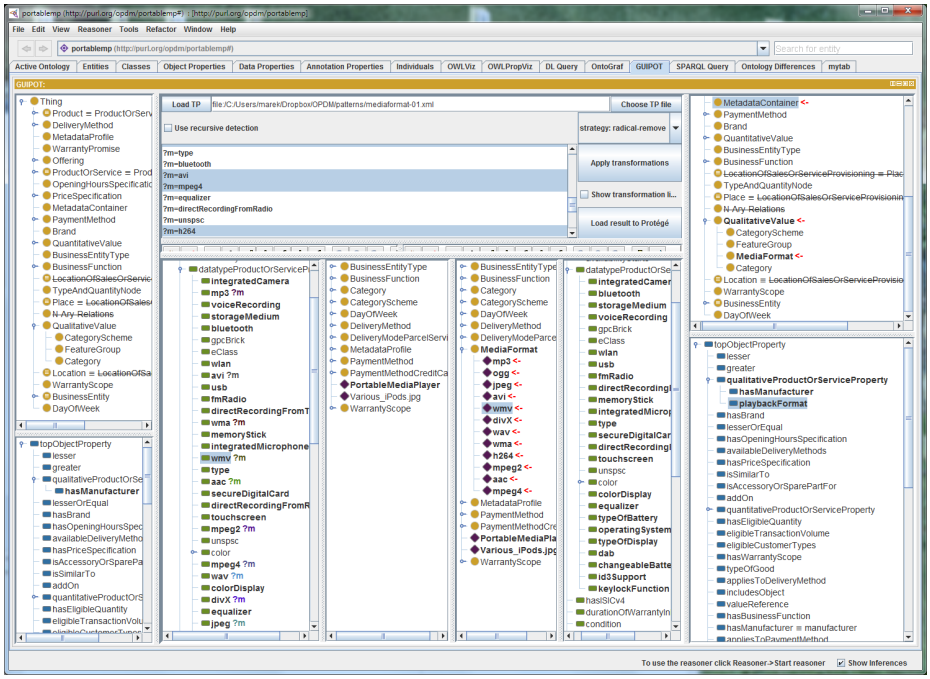

**Fig. 3.** Processing of 'boolean vs. instance' pattern by GUIPOT

## 6 Related Research

Aside the GR+OPDM ontology cluster, product ontologies have been systematically researched in a Korean national project [6]. These ontologies have been built from the beginning as heavy-weight, suited for the purpose of B2B transactions or even application in public procurement [7]. In contrast, our approach is to start with medium-complex ontologies that are easy to understand and adapt by e-commerce web masters, while further complexity can be added via semi-automated adaptation precisely defined by declarative transformation patterns.

A pattern-based method similar to PatOMat[22] is OPPL, which however doesn't allow for user interaction between the pattern detection and actual transformation phases and covers a restricted set of operations. The authors of [8] implemented an approach similar to ours, however, the transformation is considered at the data level of an ontology rather than at the schema level as (primarily) in our approach. The approach in [5] allows to transform ontologies (among other) from OWL DL, they are however directed into a generic meta-model or a different specific meta-model such as that of UML or XML Schema. PatOMat, in contrast, translates an ontology into its modelling alternatives within OWL. To the best of our knowledge, none of the alternative approaches has been applied on the product ontology domain.

[22] OPPL was actually partly reused in the first version of the framework [11].

## 7 Conclusions and Future Work

The presented research leverages on several years of research on both e-commerce ontology principles and ontology transformation techniques. It aims to provide collections of product ontologies with better internal coherence as well as external reusability, in particular, in the linked data world.

In the future, we also plan to address *import-based extrinsic incoherence*, i.e., adaptation of various legacy ontologies to GR-based modeling. Presumably, the design of ontologies for *novel domains* of products and services (such as the building industry, which plays an important role in public procurement) will also bring into light novel kinds of pattern, thus leading to enrichment of the transformation pattern library. The proliferation of specific transformation patterns will also need to be backed by a user-friendly *pattern portal* integrated with the mainstream ontology pattern portal.[23]

## References

1. Ding Y., Fensel D., Klein M., Omelayenko B., Schulten E.: The role of ontologies in e-Commerce. In: Handbook on Ontologies, Springer, 2004.
2. Heath T., Bizer C.: Linked Data: Evolving the Web into a Global Data Space (1st edition). Morgan & Claypool, 2011.
3. Hepp M.: GoodRelations: An Ontology for Describing Products and Services Offers on the Web. In: Proc. EKAW2008, 2008, Springer LNCS, Vol.5268, 332-347.
4. Klímek J., Knap T., Mynarz J., Nečaský M., Svátek V.: Framework for Creating Linked Data in the Domain of Public Sector Contracts. Deliverable 9a.1.1 of the EU FP7 LOD2 project. Online, `http://lod2.eu/Deliverable/D9a.1.1.html`
5. Kensche D., Quix C., Chatti M., Jarke M.: GeRoMe: A Generic Role Based Metamodel for Model Management. *Journal on Data Semantics*, 82–117, Vol.8, 2007.
6. Lee T., Lee I.-h., Lee S., Lee S.-g., Kim D., Chun J., Lee H., Shim J.: Building an operational product ontology system. *El. Commerce Res. and App.* 5, 16-28 (2006).
7. Lee H., Shim J., Lee S., Lee S.-g.: Modeling Considerations for Product Ontology. In: Advances in Conceptual Modeling – Theory and Practice, LNCS 4231, 2006.
8. Parreiras F. S., Staab S., Schenk S., Winter A.: In: Model Driven Specification of Ontology Translations. In: ER-2008, Springer, LNCS 5231.
9. Šváb-Zamazal O., Dudáš M., Svátek V.: User-Friedly Pattern-Based Transformation of OWL Ontologies. In: Proc. EKAW 2012, Galway, Springer-Verlag, LNCS 7603.
10. Šváb-Zamazal O., Schlicht A., Stuckenschmidt H., Svátek V.: Constructs Replacing and Complexity Downgrading via a Generic OWL Ontology Transformation Framework. In: Sofsem 2013, Springer, LNCS 7741.
11. Šváb-Zamazal O., Svátek V., Iannone L.: Pattern-Based Ontology Transformation Service Exploiting OPPL and OWL-API. In: EKAW-2010, Lisbon, Portugal, 2010.
12. Svátek V., Šváb-Zamazal O., Vacura M.: Adapting Ontologies to Content Patterns using Transformation Patterns. In: Workshop on Ontology Patterns (WOP 2010) collocated with ISWC'10, Shanghai, China, November 8, 2010. Online `http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-671/.`
13. Zamazal O., Bühmann L., Svátek V.: Checking and Repairing Ontological Naming Patterns using ORE and PatOMat. In: WoDOOM'13, workshop at ESWC 2013.

[23] `http://ontologydesignpatterns.org`