Computers & Industrial Engineering 83 (2015) 39–60

Contents lists available at ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

A framework to canonicalize manufacturing service capability models

Boonserm Kulvatunyou [a], Yunsu Lee [a,b,*], Nenad Ivezic [a], Yun Peng [b]

[a] Systems Integration Division, National Institute of Standards and Technology, Gaithersburg, MD 20899, USA
[b] Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County, Baltimore, MD 21250, USA

ARTICLE INFO

Article history:
Received 6 September 2013
Received in revised form 14 January 2015
Accepted 29 January 2015
Available online 7 February 2015

Keywords:
Manufacturing service capability
Ontology design pattern
Pattern-based ontology transformation
Canonicalization
OWL
Semantic mediation

ABSTRACT

The capability to share precisely defined information models, which reveal a supplier's *manufacturing service capability* (MSC) with anyone who needs it, is key to the creation of more agile supply chains. Today, unfortunately, this capability does not exist. Why? Because most suppliers use proprietary information models to represent and share their MSC information! This limits both the semantic precision in the models, which is needed for interoperability, and the level of agility in the supply chains. The availability of a semantically precise and rich reference MSC ontology could address both of these limitations. Based on our prior research, the development of such an ontology will require a semantic mediation process between the proprietary MSC models and the reference MSC ontology. At the heart of every known, semantic-mediation process is a mapping between a proprietary MSC model and the reference MSC ontology. Such a mapping must deal with the structural and semantic conflicts between the two. In this paper, we propose a new approach, which we call *canonicalization* to address the structural conflicts. The semantic conflicts are addressed using logical mapping. The canonicalization pre-processes the structural representations of the proprietary models and then aligns them using ontology design patterns which are also used in the reference ontology. This simplifies both the mapping problems themselves and the resulting mapping statements considerably. In the paper, we also demonstrate our approach and its benefits in the context of a description-logic-based semantic mediation using the Ontology Web Language (OWL).

1. Introduction

The Smart Manufacturing Leadership Coalition stated in its 2011 workshop report that the capability to share manufacturing service capability (MSC) information was the key to the creation of more agile and better optimized supply chains (SMLC, 2011). Min and Zhou (2002) also showed that this same capability could enable successful supply chain integration. Ameri and Dutta (2006) showed that integration was only possible when that MSC information is semantically precise, complete, and interoperable. Currently, this is not the case, because manufacturing companies provide their MSC information in proprietary MSC data models. Examples of these *proprietary MSC data models* can be found in every online marketplace dedicated to finding manufacturing suppliers for OEMs. These proprietary MSC data models are heterogeneous in their structures and representations, which make it hard for the OEMs to understand those models and find the best supplier that fits their needs. In situations like this, it is clear that information sharing, which is critical to the success of both the OEM and the supplier, is extremely difficult and costly.

Researchers have shown that a reference ontology can enhance the access to and precision of information models. In particular, Ye et al. (2007), Lu et al. (2013), Wang et al. (2013), and Zheng and Terpenny (2013) use Web Ontology Language (OWL) (W3C, 2009a) coupled with Semantic Web Rule Language (SWRL) (W3C, 2004a) to link local and reference ontologies. Kulvatunyou et al. (2013) and Tsinaraki et al. (2004) achieve similar linkages using only OWL axioms.

The *OWL-based semantic mediation* approach in Kulvatunyou et al. (2013) uses an OWL reasoner and OWL mapping axioms to inherit semantics from a semantically rich *reference MSC ontology*.[1] This approach enhances semantic precision and coverage and

* Corresponding author at: 100 Bureau Drive, Stop 8260, Gaithersburg, MD 20899-8260, USA. Tel.: +1 3019758798.
E-mail address: yslee@nist.gov (Y. Lee).

[1] In this paper the term "manufacturing service capability (MSC) model" or data model generally includes both schema and instance data. However, the reference ontology generally does not have instance data. A simple example of an instance data is 'Company A has drilling process capability with 0.025 mm precision'. We use the term 'MSC data model' in a very general sense to refer to any structured or semi-structured MSC information source; while the term 'MSC model' refers to formally encoded information specifically in OWL.

http://dx.doi.org/10.1016/j.cie.2015.01.027
0360-8352/

resolves semantic conflicts across proprietary MSC data models. The approach worked because the reference MSC ontology provided a common domain model and terminology. It has three major steps. First, information in the proprietary MSC data models is transformed into the common RDF syntax (W3C, 2004b) using the OWL semantics. Second, the resulting *OWL-encoded proprietary MSC model* is mapped to the reference ontology using OWL axioms. Finally, the description logic inference, over the OWL-encoded proprietary MSC models, the reference ontology, and the mapping axioms, results in improved MSC information sharing.

In this paper, we focus on transformation and the axioms. There are generally two ways to transform proprietary MSC data models into OWL: purely syntactic or with semantic interpretation. In the purely syntactic way, generic transformation rules are applied to the data source. Those rules are based on the underlying schema language. For example, in the case of relational databases, tables are transformed into classes and columns are transformed into properties. In the semantic-interpretation way, humans write rules that are specific to the data source schema and they use them for the transformation. Such rules are typically developed from the sole viewpoint of the data source owner.

In either case, the resulting models, called *arbitrary OWL-encoded proprietary MSC model*,[2] are not aligned structurally with the target reference ontology (see the top of Fig. 1). Such an arbitrary OWL-encoded proprietary MSC model can render the OWL mapping axioms, which are required by the approach in Kulvatunyou et al. (2013), exceedingly complex, if it is at all possible. Technologies such as the D2RQ (D2RQ, 2014) and the W3C's R2RML (Relational Database to RDF Mapping Language) (Das et al., 2012) support both the pure syntactical as well as the semantic-interpretation transformation practices.

In this paper, we propose a methodology, called a canonicalization approach, to streamline the OWL-based semantic mediation process by simplifying the OWL mapping axioms and the actual mapping itself. First, we transform the proprietary MSC data model automatically by using a common, syntactic, rule set that is independent of its source data schema. Second, a human applies a canonicalization by transforming the data again using a set of design patterns. Third, the human writes the OWL mapping axioms against the reference ontology. Since the design patterns used in the canonicalization are also used in the reference ontology, the resulting *canonicalized OWL-encoded proprietary MSC model* is more structurally aligned and, therefore, simpler to map to the reference ontology. This proposed methodology is illustrated at the bottom of Fig. 1.

In this paper, we also validate our approach by providing quantitative and qualitative analysis for a manufacturing semantic mediation example. The qualitative analysis will show that canonicalization can (1) amend a model not originally suited for semantic mediation via OWL DL, (2) simplify the mapping by avoiding the need for complex, OWL-class expressions in the mapping axioms, and (3) simplify the mapping maintenance by reducing the number of, and complexity of, mapping axioms. The quantitative analysis will show that computational time grows cubically when a certain, yet common, type of structural conflicts is resolved without canonicalization, as opposed to linearly when using canonicalization.

The rest of the paper is structured as follows. In the next section, we provide a literature review. In Section 3, we characterize canonicalization by the types of semantic conflicts it can address. Section 4 introduces the proposed canonicalization framework. It is followed with Section 5, which validates the applicability and usefulness of the framework with a running example. Section 6 presents the qualitative and quantitative analyses. Finally, we provide a conclusion and remarks on the current work and our future plans in Section 7.

[2] By arbitrary, we mean that the MSC model inconsistently and sub-optimally uses one or more approaches to express manufacturing information using the OWL language, whether it involves class-based, property-based, or some general axiomatic representation that is specific to proprietary view of the data.

2. Literature review

The importance of a reference model in semantic mediation has been emphasized in recent research. Bloomfield et al. (2012) proposed a core, manufacturing-simulation reference model to improve the data exchange between manufacturing simulations throughout the product life cycle. Wang et al. (2013) provided a shared-negotiation ontology to address communicative interoperability problems in supply chain negotiation. Zheng and Terpenny (2013) enhanced the semantics of legacy information by combining a global ontology with the legacy domain knowledge. The global ontology served as the reference model to provide additional semantics to the legacy domain knowledge. As noted above, Kulvatunyou et al. (2013) provided details of semantic mediation using the Web Ontology Language (OWL) (W3C, 2009a). In that work, the inference over the mapping between the proprietary and the semantically rich reference OWL model results in semantic enhancement to the proprietary OWL model.

All of the previously mentioned approaches require a mapping between the proprietary model and the reference model. According to Shvaiko and Euzenat (2011), they note, however, that developing such a mapping is one of the most difficult tasks in the semantic mediation, especially if there are structural conflicts. It is not surprising then that none of the previously mentioned semantic mediation approaches provides any methods or tools to assist in the mapping task. In addition, our evaluation of the ontology-mapping approaches described in Noy and Musen (2003) and McGuinness et al. (2000) found that they also do not perform well in the face of structural conflicts. Our hypothesis is that the mapping task could be simplified if the proprietary model is encoded with the same Ontology Design Patterns (ODPs) as the reference ontology. The reason is that concepts in the proprietary model would be represented with the same types of entities and with the same relationship structures used in the reference ontology. We call this, ontology canonicalization.

The approach described in Svab-Zamazal et al. (2009) and Svab-Zamazal and Svatek (2011) includes workable methods and tools for the ODP-based ontology transformations. Together, these methods and tools are called PATOMAT. PATOMAT produces a well-defined XML schema for both pattern definitions and transformation rules. In addition, the authors developed the functionality and software to generate a SPARQL query from the pattern definitions. That software uses an OPPL application interface (OPPL, 2012) for pattern transformation and a GUI editor for capturing both the source and target ontology patterns and the associated transformation rules.

PATOMAT provides a good foundation for our ontology canonicalization approach. However, several enhancements are needed. First, PATOMAT does not deal with the representative artifacts that represent the varying parts of the source ontology pattern. This means that whenever there are multiple pattern instances that use the same source ontology pattern, PATOMAT does not generate the correct, recursive, transformation rules. Second, PATOMAT does not provide any method either to generate a source ontology pattern or to retrieve a reusable target ontology pattern. This means that all the patterns must be defined manually. Lastly, PATOMAT has no facilities to deal with literal value pattern detections and transformations at present. In this paper, we describe a framework that fills these gaps in PATOMAT so that it can be used to fully canonicalize a proprietary ontological model.

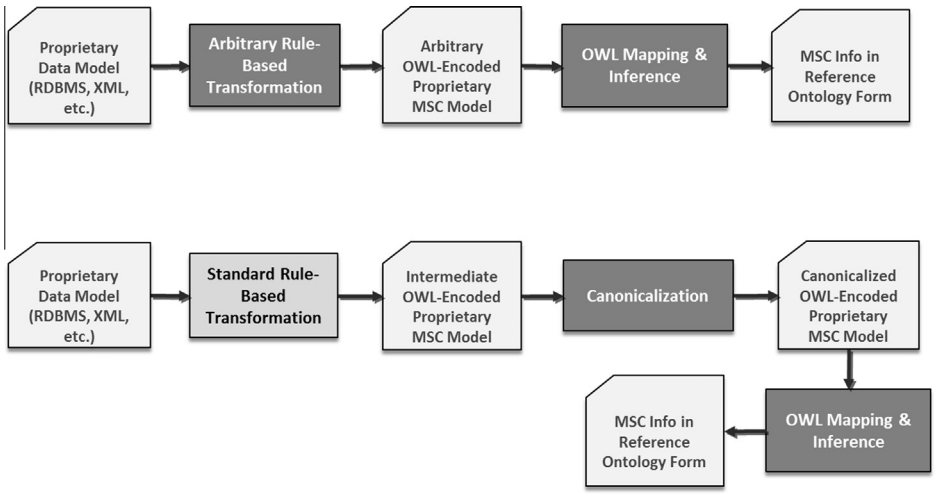


Fig. 1. Typical (top) and proposed (bottom) process to OWL-based semantic mediation between proprietary MSC data models and a Reference ontology (dark grey boxes typically require human involvement and are not fully automated).

3. Canonicalization defined

Conceptually, canonicalization is a way to preprocess diverse proprietary representations to simplify mappings. To implement this concept, we use logical and conceptual ontology design patterns. Using these patterns, as we will show, in the initial encodings of both the proprietary and reference MSC information models avoids certain conflicts in the mapping process. As shown in Sheth and Kashyap (1992) and Park and Ram (2004), these conflicts can occur at two different levels the *data-level* and *schema-level.*

3.1. Data-level conflicts

Data-level conflicts are caused by multiple representations and interpretations of similar data. In our work, we deal with three such conflicts: *data-representation* conflicts, *data-unit* conflicts, and *data-precision* conflicts. Data-representation conflicts occur when semantically equivalent values are represented differently – for example, `05/08/2012` and `May-08-2012`. The data-unit conflicts occur when the same quantities are represented with different units – for example, `2 inches` and `5 centimeters`. Data-precision conflicts occur when different scaling methods are used to represent the same concept. For example, consider the concept temperature. One way to represent temperature is to use a continuous scale from `0` to `100`; another way is to use a discrete scale like `cold`, `cool`, `warm`, and `hot`.

3.2. Schema-level conflicts

Schema level conflicts include naming conflicts, entity-identifier conflicts, schema-isomorphism conflicts, generalization conflicts, aggregation conflicts, and schematic discrepancies. Naming conflicts occur when two semantically identical concepts are named differently (synonyms); or, when two semantically different concepts are named the same (homonyms). Naming conflicts are applicable to OWL classes and properties. Entity-identifier conflicts can occur when differing primary keys are used for the same entity in different databases. This can occur in OWL when multiple class instances (individuals) with different URIs (Uniform Resource Identifiers) refer to the same individual. Isomorphism conflicts occur when two semantically identical concepts are modeled with either a different set or a different number of attributes. Consider for example `Address` which can be modeled as `Address(Line1, Line2, Zip)` and `Address(Street, City, State, Zip)`. Isomorphism conflicts are applicable to OWL classes in the sense that they can have different properties.

Generalization conflicts occur when objects/classes subsume one another – for example, `Student(ID, Name)` subsumes `GraduateStudent(ID, Name)`. Generalization conflicts are applicable to OWL classes and properties particularly when two models have different subsumption hierarchies. Aggregation conflicts occur when a property of a class is an aggregation of properties from multiple instances of another class. For example, the `MonthlyProduction(ID, Month, Year, Item, Quantity)` is an aggregation of the `DailyProduction(ID, Date, Item, Quantity)`. Schematic discrepancies occur when information is modeled using differing constructs – table name, attribute name, and attribute value. In OWL, for example, information about a supplier providing a CNC Machining Service may be modeled using (1) a class declaration axiom (a supplier is a type of `CNCMachiningService` class), (2) an object property assertion (the supplier has an object property pointing to an instance of `CNCMachiningService` class), (3) the supplier has an object property pointing to a `CNCMachiningService` instance of a `ManufacturingService` class), (4) a literal data property assertion (the supplier has a string-based property pointing to ‘‘`CNCMachiningService`’’, or (5) a boolean data property assertion (the supplier has a `isCNCMachiningServiceProvider` property with the value `true`).

3.3. What does canonicalization mean?

In our work, canonicalization means encoding the proprietary MSC data model such that it follows a set of Ontology Design Patterns (ODPs). Gangemi (2005) has described two types of ODPs: logical and conceptual. Logical ODPs are independent of domain conceptualization; conceptual ODPs, on the other hand, are specific to a domain conceptualization. Examples of logical ODPs are those given by the W3C Semantic Web Task Force on ODPs (W3C, 2005a; W3C, 2005b) including Representing Classes As Property Values on the Semantic Web and Representing Specified Values in OWL. An example of a conceptual ODP is Participation at spatio-temporal location (Gangemi, 2005).

Our research focuses mostly on conceptual ODPs. An example of a conceptual OPD related to our work is Dimensional Capability

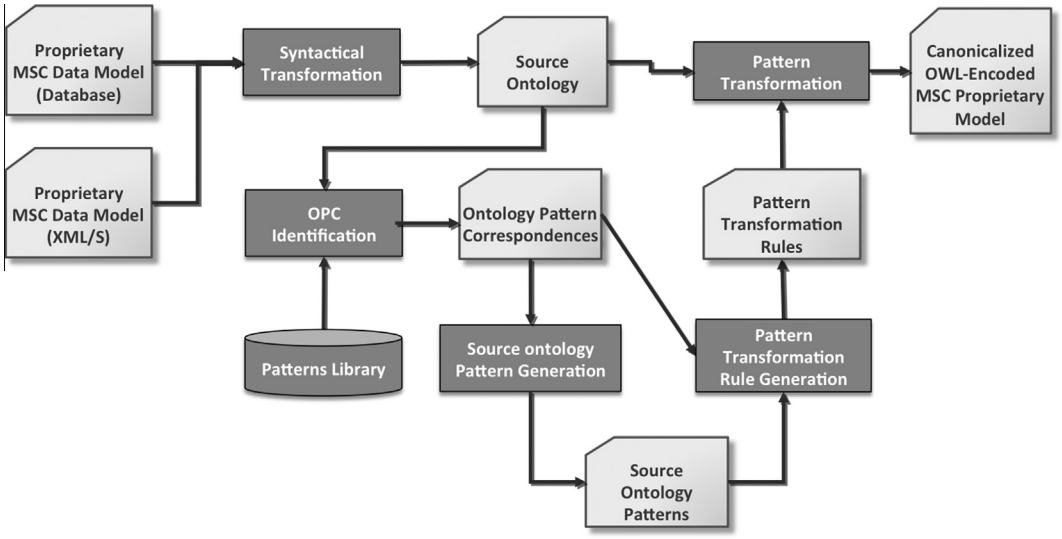


Fig. 2. Canonicalization framework.

Expression ODP. This pattern is represented as a class with two data properties: a minimum value and a maximum value. These properties effectively provide a range of permissible values. So, whenever the representation of a concept includes a range of permissible values, the Dimensional Capability Expression ODP must be used in that representation. Canonicalization identifies all of those concepts and ensures that the correct ODPs are used in their encoding.

Canonicalization resolves two of the schema-level conflicts described above: isomorphism conflicts and schematic discrepancies. Canonicalization also resolves a wide range of data-level conflicts as well. In this paper, however, we discuss the resolution of the two schema-level conflicts only.

With respect to the isomorphism conflicts, we looked at two cases: *those with the same data precision* and those *with differing data precision*.[3] The isomorphism conflicts *with the same data precision* usually occur when two different objects contain the same set of information. We extend this type of conflict to include cases where the data precisions are different provided they can be made the same by a transformation based on existing ODPs. Consider, for example, an information object called `MaximumPartSize(value) of an EDM machine` that describes the largest part that can be machined on a particular EDM machine. Since the minimum size is zero (0), we can "canonicalized" this object by using the `DimensionalCapability(minValue, maxValue)` ODP to create a new object called the `PartSizeCapability(minValue, maxValue)` where the `maxValue = value` and the `minValue = 0`. The `minValue` may be alternatively derived from common domain knowledge.

Isomorphism conflicts with differing data precision occur when two objects, which are semantically equivalent, have differing sets of information and those differences are not transformable using a set of ODPs. For example, suppose the `EDMService` object and the `EDMMachiningService` object are supposed to represent the capabilities of the same EDM machine. `EDMService` has three properties: `(ID, Type, SpecialService)`; yet `EDMMachiningService` has four properties: `(ID, Type, SpecialService, MaxPartLength)`. Canonicalization does not resolve these types of isomorphism conflicts.

3.4. Remarks

In this section, we have described the canonicalization from the perspective of schematic differences resolutions. In the next section, we describe the framework to canonicalize proprietary manufacturing service capability data models that originally may be implemented in various syntaxes such as relational databases, XML and XML schemas (W3C, 2006; W3C, 2004c).

4. Canonicalization framework

This section describes the proposed canonicalization framework. Readers are referred to the OWL 2 Structural Specification (W3C, 2009b) for definitions of a number of OWL-related terms used throughout the rest of the discussion. These terms include *entities* – various types of entities including *classes*, *datatypes*, *object properties*, *data properties*, *annotation properties*, and *named individuals*; *literals*; and *axioms* – various types of axioms including *declaration axioms*, *class axioms*, *object property axioms*, *datatype definitions*, *keys*, *assertions/facts*, and *annotation axioms*. For the purpose of our discussion, entities, literals, and axioms are collectively referred to as *ontology artifacts*.

4.1. Overview

This section describes the proposed canonicalization framework, which is outlined in Fig. 2. The initial input to the canonicalization is a proprietary MSC data model. That model can be based on different syntaxes such as relational databases, XML and XML schemas (XML databases). In the first step of canonicalization process, these heterogeneous syntaxes are transformed into a common RDF graph syntax using OWL DL vocabulary and semantics. This Syntactical Transformation is generally automatic because it uses a standard transformation rule set that is independent of the MSC information semantics but specific to the proprietary data modeling syntax. The output of this step is a *source ontology*.[4] This ontology and the current ODP library become the input to second step, which we call *Ontology Pattern Correspondence (OPC) identification*. Each ODP in the library provides the structural patterns needed

[3] Canonicalization helps avoid only the former case. The latter case may be better left to handle via the OWL's description logic-based mapping.

[4] Source ontology is mapped to the Intermediate OWL-Encoded Proprietary MSC model in Fig. 1.

to model archetypical, manufacturing-service objects and relations that are already in the source ontology.

In this OPC identification step, we determine which ODPs go with which ontology fragments in the source ontology. The output of this step is a set of OPCs. The OPCs are used to construct *source ontology patterns* in the *source ontology pattern generation* step and also to retrieve the applicable *target ontology pattern* from the pattern library. A source ontology pattern is used to retrieve all matching *pattern instances* from the source ontology. A pattern instance is a source ontology fragment to be transformed with respect to a target ontology pattern. These artifacts are transformed using the *pattern transformation rules* that are generated in the *transformation rule generation* step. The final pattern transformation step (1) applies the source ontology pattern and then (2) executes the respective pattern transformation rules on the source ontology to generate the canonicalized OWL-encoded proprietary MSC model (canonicalized proprietary MSC model for short).

Subsequent sections describe ODP and each canonicalization step in more detail.

4.2. OWL ontology design pattern

Pattern-based approaches for ontology design have been gaining popularity recently. The reason is that by reusing existing, tested patterns as building blocks, a domain ontology can be constructed quickly with higher quality and less conceptual divergence (Gangemi, 2005). A large amount of such patterns has been proposed in the ontology design community (Presutti et al., 2008).[5] The community presented a general template for describing ODPs and an initial repository of OWL-based ODPs. We define a formal representation of OWL ODPs as follows:

Definition 1: Archetypical Ontology is a fragment of OWL structure represented with abstract concepts. It is a 4-tuple {E, L, AI, A}.
- E is a set of OWL entities
- L is a set of OWL literals
- AI is a set of OWL anonymous individual
- Ax is a set of OWL axioms

Definition 2: ODP is a 2-tuple {Sig, BE}
- Sig is a non-empty set representing an ontology signature
- BE is a non-empty set representing binding expressions

Definition 3: Ontology Signature is a 2-tuple {SE, SX}
- SE is a non-empty set of entity and literal parameters
- SX is a set of axioms relating members in SE

Definition 4: Binding Expressions is a 2-tuple {SE, C}
- SE is a non-empty set of entity and literal parameters, as in Definition 2
- C ⊆ U L, is a non-empty set of concepts and values assigned to the parameters in BE that give a specific meaning to the ontology signature

---

An ODP[6] is represented by a signature and a set of binding expressions. An ODP signature is a parameterized structure of an archetypical ontology; the binding expressions connect parameters in the ODP signature to a subset of entities and literals in the archetypical ontology fragment. These entities and literals are concepts and values that give a specific meaning to the ODP signature. The entities and literals are divided into two groups: conceptual and representative. Only the conceptual entities and literals, however, are used in the binding expressions to convey the meaning of the ODP. The representative entities and literals represent the varying parts of the pattern; they must be replaced by (1) entities and literals from the source ontology or (2) by defaulted values when the ODP is used.

Figs. 3 and 4 illustrate this through a `Supplier-Service` ODP example. Fig. 3 shows the archetypical ontology fragment (conceptual pattern) of the ODP. Fig. 4 shows the ODP represented by a signature and binding expressions. With respect to the archetypical ontology fragment in Fig. 3 and ODP in Fig. 4, p:Supplier, `p:Profile`, `p:ServiceCategory`, `p:hasProfile` and `p:hasService` are conceptual. While `p:ServiceSubcategory`, `p:SupplierInstance`, `p:ProfileInstance`, and `p:CategoryInstance` are representative. Notice that only the conceptual entities are used in the binding expressions where `C1`, `C2`, `C3`, `OP1`, and `OP2` are bound (in this example there is no conceptual literal). `I1`, `I2`, `I3`, and `C4` are not bound because they are parameterized part of the ODP. It should be noted that the same ODP signature may be used by multiple ODPs but with differing binding expressions. Table 1 shows the serialization of this ODP (note that unbounded entities that are a parameterized part of the ODP simply do not have associated binding expression).

4.3. Syntactical Transformation

The Syntactical Transformation step applies a standard, rule-based transformation to convert heterogeneous syntaxes of proprietary MSC data models into the common RDF graph syntax using OWL DL vocabulary and semantics. The output of this step is called the source ontology, which corresponds to the intermediate OWL-encoded proprietary MSC model in Fig. 1.

This step can be largely automated when the proprietary MSC data model is structured information (e.g., RDB) as opposed to unstructured (e.g., text, HTML). Currently, there are many tools that support RDB-to-RDF transformations. The W3C RDB2RDF Incubator group presented a survey on these tools such as D2RQ (D2RQ, 2014), Oracle Database 11g, Virtuoso's RDF View, Metatomix Semantic Platform, RDBtoOnto, SquirrelRDF, TopBraid Composor and Triplify (Satya et al., 2009). We have investigated D2RQ in particular. It provides a mapping language to create a mapping profile between a relational database schema and RDFS/OWL ontologies. Its software platform can execute the mapping profile to create an RDF representation of a corresponding relational database (Bizer, 2003; Bizer and Seaborne, 2004; D2RQ, 2014).

In our work, the proprietary MSC data models are captured in relational databases. The standard rule-based transformation follows the default mapping profile[7] proposed in the D2RQ with minor enhancements. Since the D2RQ's default mapping profile only uses RDF vocabulary and semantics, we can enhance the mapping profile by (1) specializing RDF vocabulary with OWL vocabulary (e.g., replace `rdf:Class` with `owl:Class`) and (2) specifically replacing the generic RDF property with OWL data or object property. The resulting mapping profile is summarized below. The mapping profile uses OWL DL vocabulary and semantics without a need to be tailored to specific database entities. Hence, it can transform any relational data independent of its schema.

[5] http://www.ontologydesignpatterns.org.

[6] The requirement for canonicalization is that the reference ontology follows a set of ODPs. However, rationalization and development of ODPs is out of the scope of this paper. The design of an ODP is concerned with the semantics and associated structure to convey the semantics. The related work section points to works concerned with ODP developments and other ODPs' information that may be of interest to store within the pattern library. In this paper, ODPs are derived from the Manufacturing Description Language ontology (Ameri and Dutta, 2006).

[7] In the D2RQ framework, mapping profile is used to generate RDB-to-RDF mapping for a database schema of a relational database. The mapping is then used to execute the RDB-to-RDF transformation on a database instance using that schema.

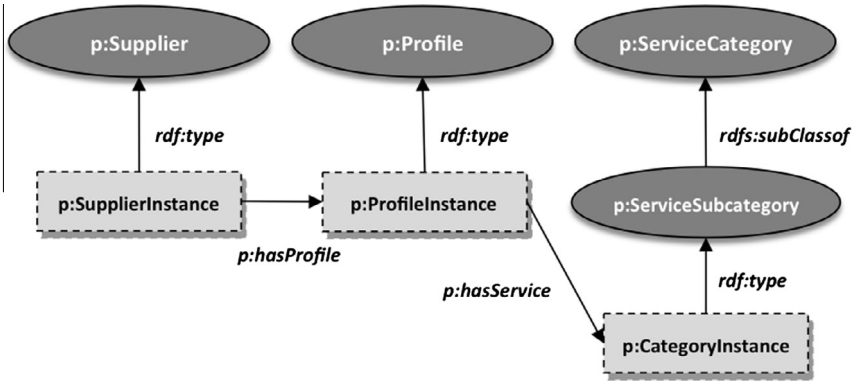


Fig. 3. Archetypical ontology of the Supplier-Service ODP.

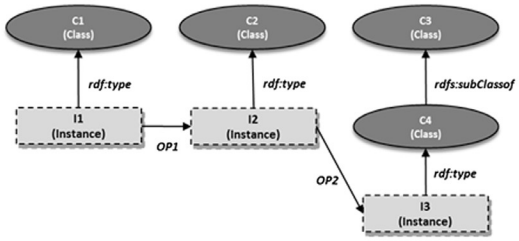
ODP Name/Description	ODP Signature	Binding Expression
Supplier-service: Supplier declares a service it provides and models a semantic service category as a (descendant) subclass of the abstract ServiceCategory class		C1 = p:Supplier C2 = p:Profile C3 = p:ServiceCategory OP1 = p:hasProfile OP2 = p:hasService **Unbounded Entities:** C4, I1, I2, I3

Fig. 4. Supplier-Service ODP example.

Relational database to OWL DL mapping profile:

1. A table is mapped as an `owl:Class` (class declaration).
2. A record in the RDB is mapped as an `owl:NamedIndividual` (class assertion) of the corresponding class.
3. An attribute that is not a foreign key, is mapped to an `owl:DataProperty` (data property declaration); and its value is mapped as a data property assertion whose literals have data types carried from the database schema.
4. An attribute that is a foreign key attribute is mapped to an `owl:ObjectProperty` (object property declaration); and its value is mapped as an object property assertion.

Fig. 5 below shows an example of the transformation from a relational database table into an OWL DL source ontology using the above mapping profile. The Capability table is converted into an `owl:Class` named `Capability`. The `ID` attribute is converted into `owl:DataProperty` named `ID`. The `Capability_Name` attribute is converted into an `owl:DataProperty`, named `Capability_Name`. The record, which has the value `5` as its key, is converted into an `owl:NamedIndividual` named `Capability/5`. Its `ID` attribute value `5` is an `xsd:integer` value of the `ID` data property. Its `Capability_Name` attribute value `CP_EDM` is an `xsd:string` value of the `Capability_Name` data property.

4.4. OPC identification

The purpose of the Ontology Pattern Correspondence (OPC) Identification step is to select a unique ODP for a specific fragment of the source ontology. The OPC identification process starts with establishing semantic links between entities/literals in the source ontology and the elements of the pattern library (by matching on their intended meaning).

Semantic links form the foundation of an ontology mapping. These links can be *homogeneous* or *heterogeneous*. A homogeneous link is between the same types of entities and literals. Examples include class-to-class, property-to-property, individual-to-individual, and literal-to-literal. A heterogeneous link is between different types of entities and literals. Examples include class-to-individual, class-to-property and property-to-individual. There is no restriction on the cardinality of either type of link. For instance, a part size capability may be represented as a single concept with the literal value ‘‘`5–25 cm`’’ on the proprietary side. The same capability, however, can be represented as multiple concepts such as minimum part size and maximum part size in the reference ontology. Solution to this situation would need 1:n semantic links.

Semantic links may be established manually or with assistance from an ontology matching algorithm (Shvaiko and Euzenat, 2011). Once established, these links provide the foundation for selecting an ODP for a specific collection of source ontology artifacts. The selected ODP is called a target ontology pattern. Note that an individual ODP can be the target pattern for several collections of source artifacts. We call the selected collection of these source artifacts, *representative ontology artifacts*. The goal of the OPC identification process is to find all of these artifacts.

Typically, semantic links identify only part of the representative ontology artifacts for an ODP. The user will need to manually supply the rest. After identify all the representative ontology artifacts for an ODP, structural differences between the representative

Table 1
Serialization of the `Supplier-Service` ODP in.

Entity					Literal	Axiom	Binding expression
Class	Indi vidual	Object property	Data property	Data type			
C1	I1	OP1	–	–	–	(I1, rdf:type, C1)	
C2	I2	OP2				(I2, rdf:type, C2)	C1 = p:Supplier
C3	I3					(C4, rdfs:subClassOf, C3)	C2 = p:Profile
C4						(I3, rdf:type, C4)	C3 = p:ServiceCategory
						(I1, OP1, I2)	OP1 = p:hasProfile
						(I2, OP2, I3)	OP2 = p:hasService

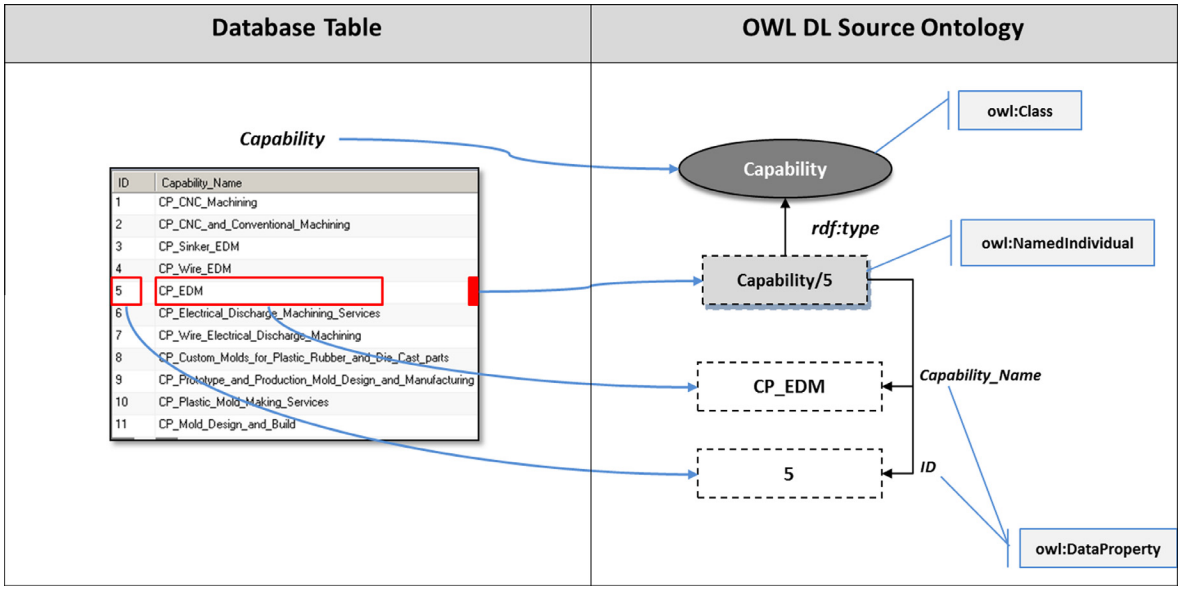


Fig. 5. Transformation example.

ontology artifacts and the ODP in each OPC are identified. If there are structure differences, the canonicalization process proceeds to rectify the differences for that OPC. Note, the identification process is not always linear because interactions with other ODPs can occur after the initial pattern transformation.

4.4.1. OPCs identification illustration

Fig. 6 shows an example of the source ontology. Fig. 7 shows some semantic links between the source ontology and the pattern library established by the user. Note that the prefix 's' denotes entities and literals from the source ontology and the prefix 'p' denotes entities and literals from the pattern library.

```
{s:Supplier, p:Supplier, Class-to-Class},
  {s:ServiceCategory, p:ServiceCategory, Class-
  to-Class}
```

Based on the matching between concepts in the semantic links and concepts in the binding expressions of ODPs, candidate ODPs are suggested as illustrated in Fig. 7. In this example, only the `supplier-service` ODP is suggested. If multiple ODPs were suggested, the user would inspect each of the suggested ODPs and their semantic relationships with the related source ontology artifacts (bottom part of Fig. 7). The canonicalization system can assist the user during this semantic relations inspection by identifying paths between the matching source ontology artifacts (in this case the `s:Supplier` and `s:ServiceCategory`). Once the user selects a suggested ODP, an ontology pattern correspondence is initiated to capture information related to the suggestion. Table 2 shows the initialization of this example as `OPC1`. `OPC1` is not yet complete as only the representative artifacts identified by the semantic links are captured. The representative artifacts do not form a complete source ontology (graph) fragment since the `s:Supplier` and `s:ServiceCategory` are not connected.

The user identifies the rest of representative source ontology artifacts for `OPC1`. The complete `OPC1` is shown in Table 3. Notice that only `s:SupplierA` and `s:EDM` instances are identified as representative source ontology artifacts. Notice also, other instances, such as `s:SupplierB` and `s:Machining` (see Fig. 6), also match this pattern. These other instances will be identified via the source ontology pattern created in the next canonicalization step. Next

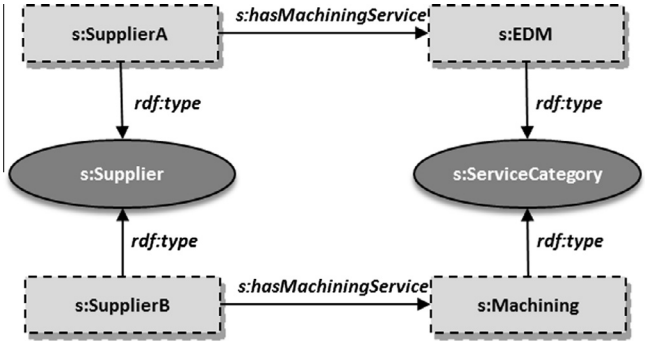


Fig. 6. An example source ontology.

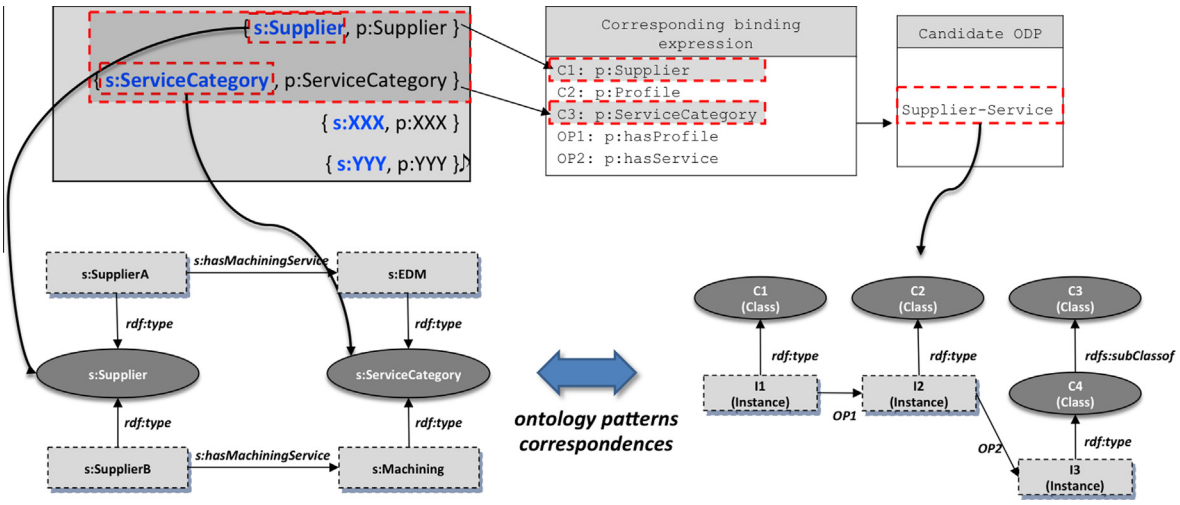


Fig. 7. OPC identification process.

Table 2
Initial state of `OPC1`.

OPC ID	Source ontology pattern		Target ontology pattern
	Artifact type	Representative artifact	

`OPC1`	`Class`
	`Individual`
	`ObjectProperty`
	`DataProperty`
	`Datatype`
	`Literalal`
	`Axiom`
		`s:Supplier, s:ServiceCategory`	`Supplier-Service`

Table 3
Finished state of `OPC1`.

OPC ID	Source ontology pattern		Target ontology pattern
	Artifact type	Representative artifact	

`OPC1`	`Class`
	`Individual`
	`ObjectProperty`
	`DataProperty`
	`Datatype`
	`Literalal`
	`Axiom`
		`s:Supplier, s:ServiceCategory`
		`s:SupplierA,s:EDM`
		`s:hasMachiningService`
		
		`(s:SupplierA, s:hasMachiningService, s:EDM)`
			`Supplier-Service`

the user determines whether `OPC1` requires subsequent canonicalization steps.

By analyzing the graph structures, the user can determine that `OPC1` does requires subsequent canonicalization steps because the logical structure of source ontology artifacts is different from that in the `Supplier-Service` ODP. The reasons are (1) schematic discrepancy, i.e., the source ontology represents the semantic service category `s:EDM` as an instance of the `s:ServiceCategory` class while the `Supplier-Service` ODP represents any semantic service category (`p:ServiceSubcategory`) as a subclass of the `p:ServiceCategory` class and (2) isomorphism conflict, e.g., `s:SupplierA` has a direct connection to the `s:EDM` service category via the `s:hasMachiningService` object property in the source ontology while the `p:SupplierInstance` has an indirect connection to the `p:CategoryInstance` through the `p:hasProfile`, `p:ProfileInstance`, and `p:hasService`.

In this step, we are only concerned with terminological links. Manually establishing these links can become cumbersome when the source ontology and pattern library are large. Since we are not dealing with structural relationships here, we believe that existing ontology matching algorithms could be quite useful. This, however, will be the topic of our future work.

4.5. Source ontology pattern generation

A source ontology pattern is used to retrieve all pattern instances that are related to an OPC from the source ontology. Each pattern instance is a source ontology fragment. Each of such instances will contain source ontology artifacts that will be transformed according to the target ontology pattern in the OPC. Source ontology patterns also have a signature and a set of binding expressions. The signature is the parameterized ontology structure: it has a fixed part and a variable part. Binding expressions define the fixed part by connecting parameters in the signature to the entities and literals in the source ontology artifacts. The remaining unbound parameters become the variable parts of the ontology structure. Together the signature and binding expressions must be sufficient enough to be converted into a query that retrieves a

collection of variable parts of the ontology structure. Each member of the collection, together with the fixed part of the source ontology pattern, makes up a pattern instance. The query must retrieve all pattern instances related to the OPC from the source ontology. At the end of the source ontology pattern generation step, a source ontology pattern is defined for each OPC output from the OPC identification step.

The source ontology pattern generation step can be largely automated. Based on the representative source ontology artifact identified in the OPC, the source ontology signature can be derived. All possible binding expressions can be generated. Specific binding expressions can be recommended based on the semantic links established earlier. That is, the framework will also suggest fixing the parameters in the source ontology signature that are linked to the fixed part of the target ontology pattern. The query generation can be automated based on the suggestion in Svab-Zamazal et al. (2009) and Svab-Zamazal and Svatek (2011).

Fig. 8 shows a graphical representation of an exemplary source ontology pattern, called SP1. Table 4 shows its serialization. It is a pattern devised for the representative source ontology artifacts in OPC1 shown in Table 3 above. In this example, the binding expressions indicate that C1, C2, and OP1 are bound to constants, while I1 and I2 are unbound. That is, I1 and I2 will be used to retrieve all instances of the classes Supplier and ServiceCategory matching this pattern.

Fig. 9 shows the SPARQL query (W3C, 2008) automatically constructed using the source ontology pattern definition. Unbound variables I1 and I2 are outputs of the query, while axioms and binding expressions make up the condition (i.e., WHERE clause) of the query.

4.6. Pattern transformation rule generation

In this process, a *pattern transformation rule* set (PTRS) is generated for each OPC. A PTRS consists of *pattern transformation rules* (PTRs). A PTR specifies relations between parameters in the source and target ontology patterns within a particular OPC. These relations describe how the source ontology pattern should be transformed according to the target ontology pattern. Each relation in a PTR is a 3-tuple including source column, target column, and transformation expression column. The source column indicates one or more entities or literals in the source ontology pattern that will be transformed using a parameter from the pattern signature. The target column indicates the entity or literal in the target ontology pattern into which the source column will be transformed using a parameter from the pattern signature. Either the source column or target column can be null but not both. The last column, transformation expression column, indicates the specific names/IRIs to be used for the target in the output. The value can be a parameter from the target ontology pattern signature, a fixed value, or a string expression.

Transformation Type-1: same artifact type transformation (e.g., Class-to-Class, Instance-to-Instance)
Transformation Type-2: different artifact types transformation and n:1 transformation (e.g., Class-to-Instance, Property-to-Class, Classes-to-Class)
Transformation Type-3: Artifact removal transformation (e.g., source ontology pattern signature has a class A that does not have a correspondence in the target ontology pattern signature entity)
Transformation Type-4: Artifact creation transformation (e.g., source ontology pattern signature does not have the class A which is defined in the target ontology pattern)

Each PTR can be one of the three transformation types as listed below. Transformation types can be automatically determined based on source and target columns and handled automatically by the canonicalization infrastructure.

Similar to the OPC identification step, the pattern transformation rule generation step may be done manually or with assistance from an ontology matching algorithm. However, in this case the scope of the match is more specific to only ontology artifacts in the representative source ontology and in the archetypical ontology of the target ontology pattern.

Table 5 below shows an example transformation rule set, PTRS1, for OPC1. OPC1 consists of the source and target ontology pattern shown in Figs. 8 and 4. It requires 10 PTRs.

4.7. Pattern transformation

Pattern transformation executes pattern transformation rules on the source ontology. The resulting transformation is the OWL DL encoding of the canonicalized OWL-encoded proprietary MSC model (*canonicalized proprietary MSC model* for short). The

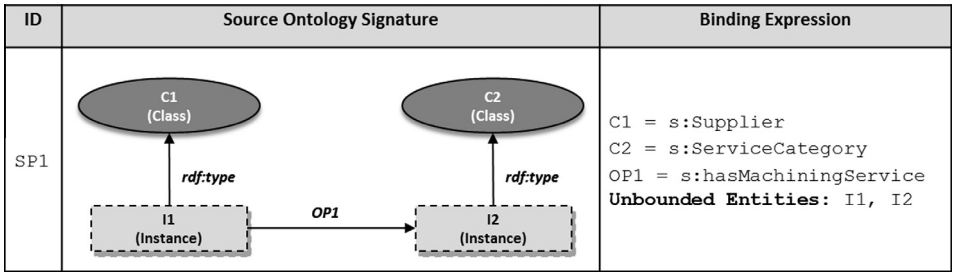


Fig. 8. Graphical representation of the source ontology pattern for OPC1.

Table 4
Serialization of the source ontology pattern SP1 in Fig. 8.

Entity					Literal	Axiom	Binding expression
Class	Individuals	Object Property	Data Property	Data type			
C1,C2	I1, I2	OP1	–	–	–	(I1, rdf:type, C1) (I2, rdf:type, C2) (I1, OP1, I2)	C1 = S:Supplier C2 = S:ServiceCategory OP1 = s:hasMachiningService

```
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX s: <http://www.nist.gov/el/sid/msnm/PortalB.owl#>

SELECT ?I1, ?I2
    WHERE {
        ?I1 rdf:type s:Supplier .
        ?I2 rdf:type s:ServiceCategory .
        ?I1 s:hasMachiningService ?I2 .
}
```

Fig. 9. Source ontology pattern SPARQL query.

Table 5
An exemplary PTRS1 for OPC1.

PTR ID	Source ontology pattern	Target ontology pattern	Transformation type	Transformation expression
PTR1.1	s:C1	p:C1	Type-1	s:C1
PTR1.2	s:C2	p:C3	Type-1	s:C2
PTR1.3	s:I1	p:I1	Type-1	s:I1
PTR1.4	s:I2	p:C4	Type-2	s:I2
PTR1.5	s:OP1	–	Type-3	–
PTR1.6	–	p:C2	Type-4	p:C2
PTR1.7	–	p:I2	Type-4	p:I2
PTR1.8	–	p:I3	Type-4	p:I3
PTR1.9	–	p:OP1	Type-4	p:OP1
PTR1.10	–	p:OP2	Type-4	p:OP2

canonicalized proprietary MSC model will be aligned structurally with an OWL DL-based reference MSC ontology that is also constructed based on the same pattern library. The pattern transformation of an OPC is divided into two sub-processes: detecting pattern instances and applying pattern transformation rules. The whole process can be automated based on the aforementioned PATOMAT work.

The detection process applies the source ontology pattern to find all pattern instances. A pattern instance is a set of source ontology's entities and literals to be transformed by a transformation rule. For example, the SPARQL query generated from the source ontology pattern shown in Fig. 9 will retrieve all pattern instances for OPC1. Two pattern instances should be returned for the source ontology in Fig. 6.

The application process applies the pattern transformation rules on the retrieved source ontology's entities and literals in the pattern instances. The output entities and literals provide all the necessary elements to establish the set of axioms in the target ontology pattern. After the pattern transformation executes all the pattern transformation rules on the source ontology, the canonicalized proprietary MSC model is obtained as the final output.

In the next section we demonstrate our canonicalization framework on a manufacturing service capability example.

5. Canonicalization example

We first describe inputs to the canonicalization process including a pattern library and a proprietary MSC data model that is captured in a relational database. We then walk through each process step in the canonicalization framework. After obtaining the canonicalized proprietary MSC model, we will show how it simplifies the mappings to the reference MSC ontology in Section 6.

5.1. Pattern library

In this example, we assume a hypothetical pattern library, which consists of four ODPs including Supplier-Service, Service-LengthCapability, Service-Categorization, and LengthCapability. Fig. 10 illustrates the definitions of these ODPs.

5.2. Proprietary MSC data model

Fig. 11 below shows a set of relational tables in a proprietary MSC data model. It represents how the supplier, service, service category, and part length capability concepts are related in that model.

5.3. Syntactical Transformation

Transformation of a table without a foreign key into OWL DL source ontology involves creating only a class, named individuals, data properties, and assertions. Fig. 12 below illustrates this transformation using the Supplier table. The Supplier table is converted into an owl:Class named s:Supplier. The record, which has the value Supplier_3 as its ID, is converted into an owl:NamedIndividual named s:Supplier_3. Its ID attribute value Supplier_3 becomes an xsd:string value of the s:Supplier_ID data property.

Transformation of a table with foreign keys involves creating a class, named individuals, object properties, data properties, and assertions. Fig. 13 illustrates the transformation using one of the records in the SupplierService table. The table is converted into an owl:Class named s:SupplierService. The record, which has 5 as its ID, is converted into owl:NamedIndividual named s:SS_3_4. Its ID attribute value 5 is an xsd:integer value of the s:SupplierService_ID data property. This table has two foreign key attributes including SupplierID and ServiceID, which are respectively primary keys of the Supplier table and Service table. These two foreign key attributes are converted into two owl:ObjectProperty declarations, namely s:SupplierService_SupplierID and s:SupplierService_ServiceID. These object properties are used to connect the s:SupplierService individuals to owl:NamedIndividual converted from the

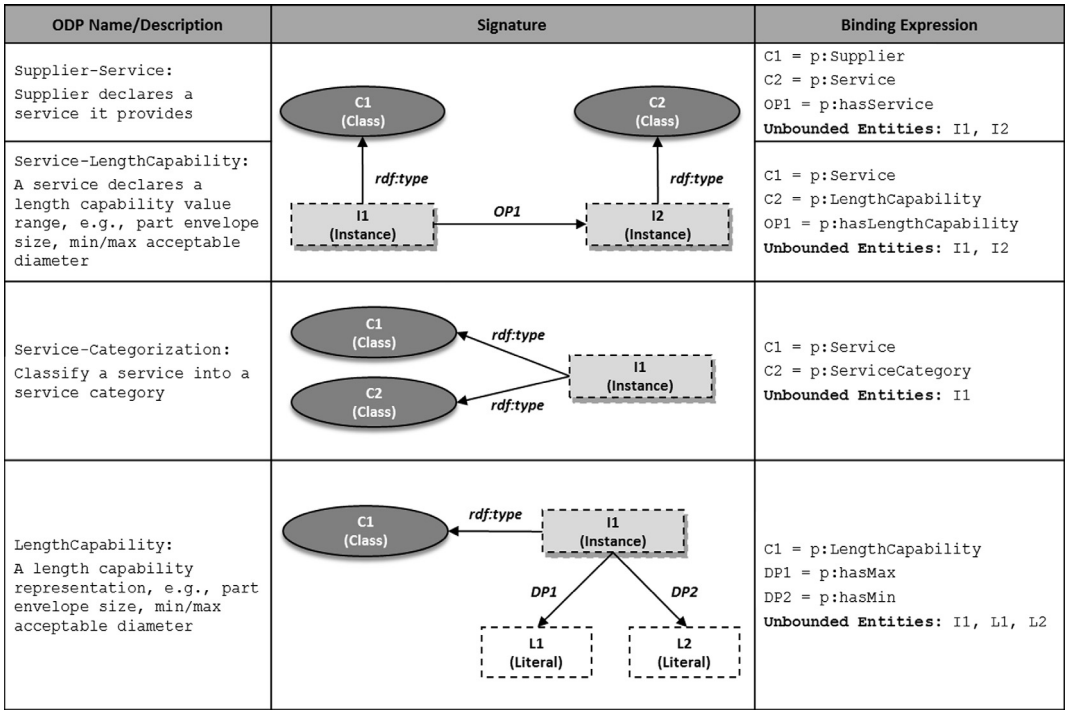


Fig. 10. Pattern library.

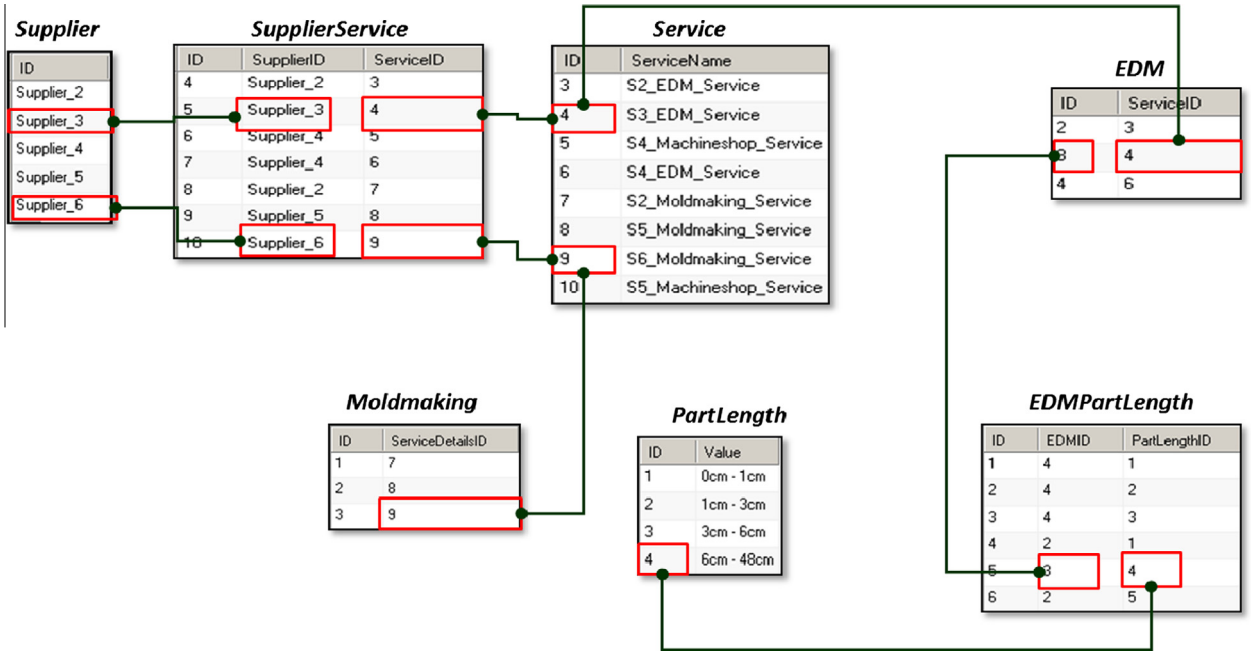


Fig. 11. Proprietary manufacturing service capability data model.

records in the `Supplier` and `Service` tables as shown in the figure.

Fig. 14 shows the source ontology, which is the output from the transformation of the proprietary MSC data model. The figure includes transformation of `s:Supplier_3` and `s:Supplier_6` records in the `Supplier` table and related records in other tables as highlighted in Fig. 11. The rest of the canonicalization illustration will be based on this data.

5.4. OPC identification

Recall that the first step in the OPC identification process is for the user to establish the semantic links between the source ontology and the pattern library (see the box below). For example, the user has linked `s:EDM` and `s:Moldmaking` with the `p:ServiceCategory`. Next, the ODPs' binding expressions are used to retrieve the ODPs that are related to these semantic links and OPCs

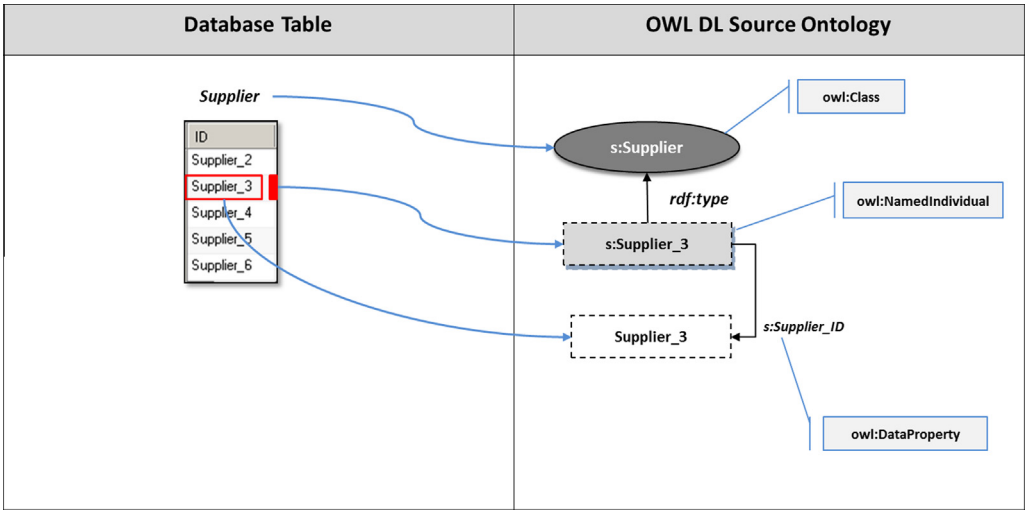


Fig. 12. Standard rule-based OWL DL encoding of the `supplier` table.

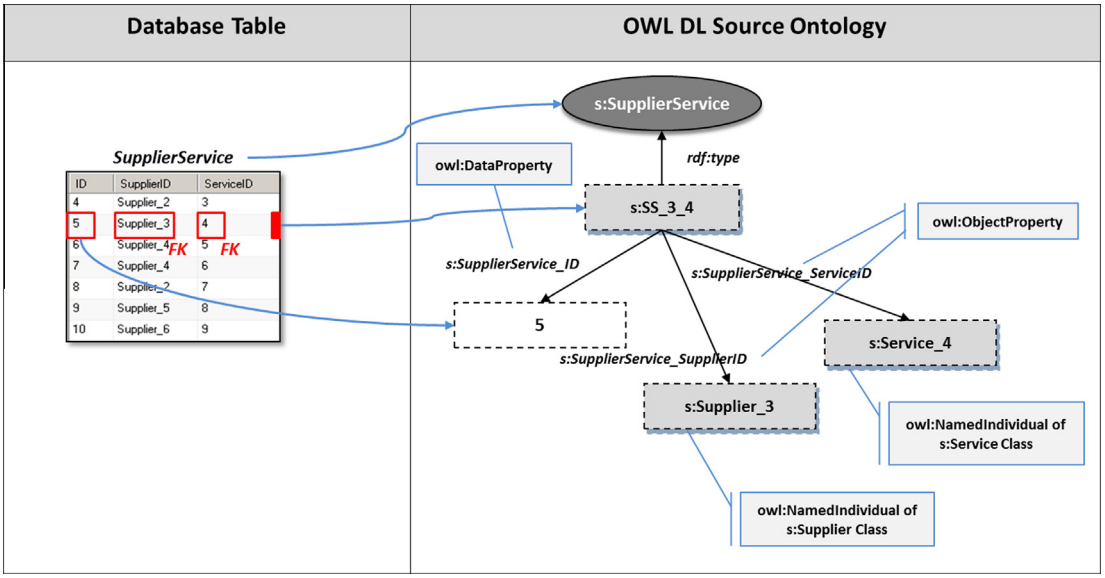


Fig. 13. Standard rule-based OWL DL encoding of the `SupplierService` table.

are initialized as shown in Table 6. Note that `OPC3` is initialized using not just the semantic links but also some logical inferences. This happened because `s:EDM` is linked to `p:ServiceCategory` and not to `p:Service` as in the `Service-LengthCapability`'s binding expression. However, as shown in Fig. 10, the `Service-Categorization` ODP illustrates that an instance of `p:Service` is an instance of `p:ServiceCategory` as well. Thus, `s:EDM` is linked indirectly to the `p:Service`; and the `OPC3` can be initialized even though `s:EDM` has no direct semantic link to `p:Service`.

```
{s:Supplier, p:Supplier, Class-to-Class},
  {s:Service, p:Service, Class-to-Class}, {s:EDM,
  p:ServiceCategory, Class-to-Class},
  {s:PartLength, p:LengthCapability, Class-to-
  Class}, {s:Moldmaking, p:ServiceCategory, Class-
  to-Class}
```

In the next step, all the representative source ontology artifacts are identified for the OPCs. The results are shown in Table 7. At this point we can identify how the logical structure of each source ontology artifact differs from its corresponding ODP. Thus, `OPC1` to `OPC5` are the OPCs output from the OPC identification on which subsequent canonicalization steps will be performed.

5.5. Source ontology pattern generation

With all the representative source ontology artifacts identified in the OPCs, source ontology patterns can be generated. Fig. 15 shows a graphical representation of the source ontology patterns based on the source ontology artifacts in Table 7.

5.6. Pattern transformation rule generation

With the source and target ontology patterns captured in the OPCs, pattern transformation rules (PTRs) can be created for each OPC. Fig. 16 visualizes the PTR relationships between the source ontology pattern `SP1` and the target ontology pattern `Supplier-Service` ODP in `OPC1`. Table 8 shows the details. `PTR1.1` and `PTR1.2` state that `s:C1` and `s:C3` in the `SP1` should be respectively transformed into the same artifact type `p:C1` and `p:C2` in the `Supplier-Service` ODP; and hence, they are type-1 transformations.

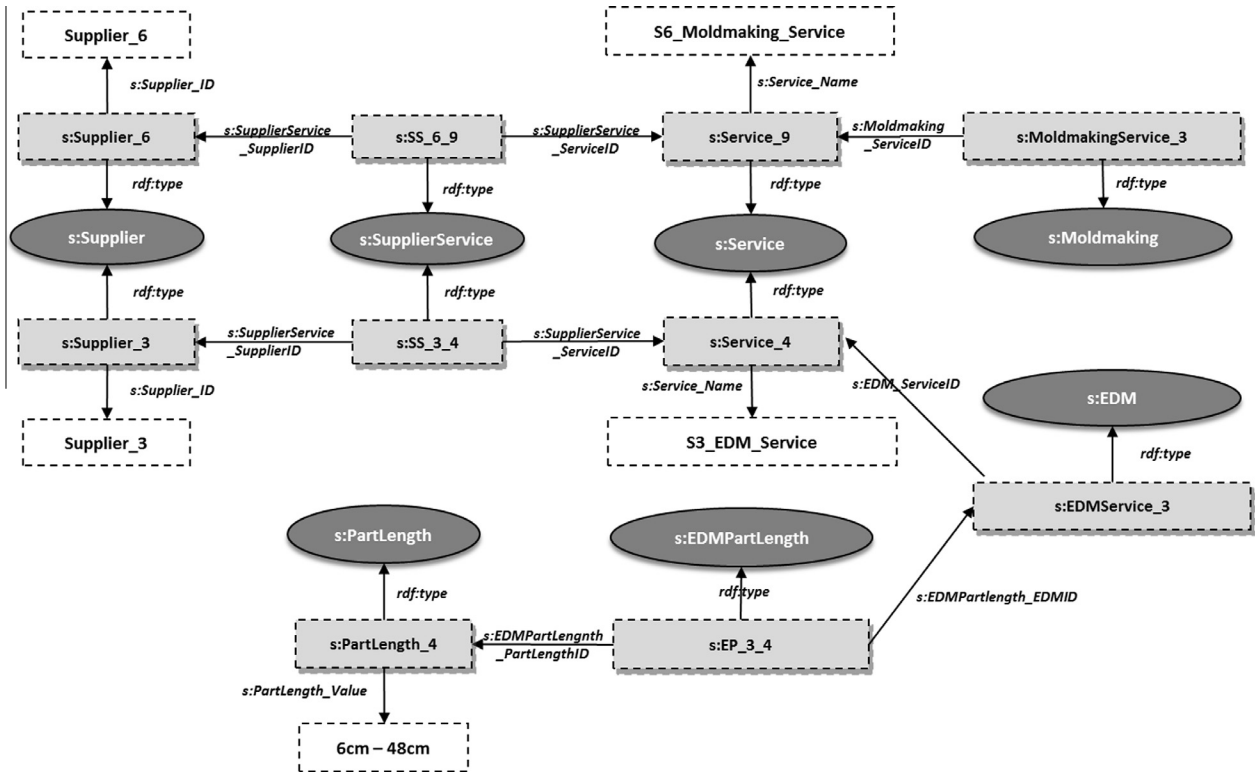


Fig. 14. Source ontology.

Table 6
Initial ontology pattern correspondences.

OPC ID	Source ontology pattern		Target ontology pattern
	Artifact type	Source ontology artifact	
OPC1	Class	s:Supplier, s:Service	Supplier-Service
OPC2	Class	s:Service,s:EDM	Service-Categorization
OPC3	Class	s:EDM, s:PartLength	Service-LengthCapability
OPC4	Class	s:PartLength	LengthCapability
OPC5	Class	s:Service, s:Moldmaking	Service-Categorization

PTR1.3 and PTR1.4 state that s:L1 and s:L2 in the SP1 on the other hand should be respectively transformed into differing artifact types p:I1 and p:I2 in the Supplier-Service ODP; and hence, they are type-2 transformations. PTR1.1 to PTR1.4 use the names/IRIs from the source ontology. Type-4 transformation is needed in PTR1.5 to create the relationship between p:I1 and p:I2 using the new object property p:OP1. Since type-4 transformation creates a new artifact and there is no source ontology entity corresponding to p:OP1, the name p:hasService from the ODP is used as shown in the Transformation Expression column. Lastly, PTR1.6 to PTR1.12 removes the unwanted artifacts with the type-3 transformation. Associated axioms can also be automatically removed.

Pattern transformation rule set for OPC2, OPC3 and OPC4 are denoted by PTRS2, PTRS3, and PTRS4, respectively. Their PTRs only need to deal with structural pattern detections and entity transformations that are similar to those of OPC1; and, consequently, can be created in the same way. The PTRs for OPC5, however, have an additional problem: they must also deal with literal values. This is because the LengthCapability ODP has two data properties: p:hasMin and p:hasMax. However, the SP5 source ontology pattern has only one data property that represents the same information in a *literal value* – ''6cm – 48cm'', for example. Fig. 17 illustrates the situation.

To deal with this situation, we define a *literal value pattern* with the following regular expression.

- ([0 – 9]+)–([0 – 9]+)cm

The first group in the regular expression, which is embraced by the first set of parentheses, corresponds to the minimum part length in the literal value and is assigned to the variable s:G1. The second group in the regular expression, which is embraced by the second set of parentheses, corresponds to the maximum part length in the literal value and is assigned to the variable s:G2. This literal value pattern detection is used in PTR5.3 and PTR5.4 as indicated by their usages of s:G1 and s:G2 in Table 9.

5.7. Pattern transformation

The pattern transformation process is a final step of the canonicalization and it executes pattern transformation rules on the source ontology for each OPC. Table 10 below summarizes the current state of the OPCs that provide sufficient information to execute the pattern transformation.

Below we illustrate the pattern transformation on OPC1, which is divided into two sub-processes, the pattern instances detection and pattern transformation rules application, as follows.

Table 7
Source and target ontology pattern correspondences output from the OPC identification.

OPC ID	Source ontology pattern		Target ontology pattern
	Artifact type	Source ontology artifact	
OPC1	Class	s:Supplier, s:SuplierService, s:Service	Supplier-Service
	Individual	s:Supplier_3, s:SS_3_4, s:Service_4	
	Object Property	s:SupplierService_SupplierID,s:SupplierService_ServiceID	
	Data Property	s:Supplier_ID, s:Service_Name	
	Datatype	xsd:string	
	Literalal	''Supplier_3'', ''S3_EDM_Service''	
	Axiom	(s:Supplier_3, rdf:type, s:Supplier), (s:SS_3_4, rdf:type, s:SuplierService), (s:SS_3_4, s:SupplierService_SupplierID, s:Supplier_3), (s:SS_3_4, s:SupplierService_ServiceID, s:Service_4), (s:Supplier_3, s:Supplier_ID, ''Supplier_3''), (s:Service_4, s:Service_Name, ''S3_EDM_Service'')	
OPC2	Class	s:Service, s:EDM	Service-Categorization
	Individual	s:Service_4, s:EDMService_3	
	Object Property	s:EDM_ServiceID	
	Data Property	s:Service_Name	
	Datatype	xsd:string	
	Literalal	''S3_EDM_Service''	
	Axiom	(s:Service_4, rdf:type, s:Service), (s:EDMService_3, rdf:type, s:EDM), (s:Service_4, s:Service_Name, ''S3_EDM_Service'') (s:EDMService_3, s:EDM_ServiceID, s:Service_4)	
OPC3	Class	s:Service, s:Moldmaking	Service-Categorization
	Individual	s:Service_9, s:MoldmakingService_3	
	Object Property	s:Moldmaking_ServiceID	
	Data Property	s:Service_Name	
	Datatype	xsd:string	
	Literalal	''S6_Moldmaking_Service''	
	Axiom	(s:Service_9, rdf:type, s:Service), (s:Service_9, s:Service_Name, ''S6_Moldmaking_Service''), (s:MoldmakingService_3, rdf:type, s:Moldmaking), (s:MoldmakingService_3, s:Moldmaking_ServiceID, s:Service_9)	
OPC4	Class	s:EDM, s:PartLength, s:EDMPartLength	Service-Length Capability
	Individual	s:EDMService_3, s:EP_3_4, s:PartLength_4	
	Object Property	s:EDMPartlength_EDMID, s:EDMPartLengnth_PartLengthID	
	Data Property	s:PartLength_Value	
	Datatype	xsd:string	
	Literalal	''6–48 cm''	
	Axiom	(s:EDMService_3, rdf:type, s:EDM), (s:PartLength_4, rdf:type, s:PartLength), (s:EP_3_4, rdf:type, s:EDMPartLength), (s:EP_3_4, s:EDMPartLengnth_PartLengthID,s:PartLength_4), (s:EP_3_4, s:EDMPartlength_EDMID, s:EDMService_3)	
OPC5	Class	s:PartLength	Length Capability
	Individual	s:PartLength_4	
	Object Property	–	
	Data Property	s:PartLength_Value	
	Datatype	xsd:string	
	Literalal	''6–48 cm''	
	Axiom	(s:PartLength_4, rdf:type, s:PartLength), (s:PartLength_4, s:PartLength_Value, ''6 cm – 48 cm'')	

The pattern instances detection sub-process uses the SPARQL query generated from the source ontology pattern SP1 to find instances of a source ontology pattern to be transformed. The SPARQL query generated from SP1 is shown in Fig. 18. It retrieves all the pattern instances, which contain source ontology entities and literals as shown in Table 11. Two pattern instances are returned in this example, based on the source ontology in Fig. 14.

The pattern transformation rules application sub-process applies PTRs on the retrieved entities and literals. Table 12 shows the result of applying PTRs in PTRS1 (Table 8) on the SPI1.1 query result shown in Table 11 (note that some SP1 variables are bound to constants and are not shown in Table 11). Fig. 19 shows the final output of the canonicalization process from executing pattern transformation on OPC1 to OPC5.

6. Analysis

This section provides qualitative and quantitative analyses to demonstrate that our canonicalization approach enables simpler OWL mapping axioms, shorter reasoning time, and better semantic mediation results. Specifically, that approach significantly improves the *OWL Mapping & Inference* step of the semantic mediation process shown in Fig. 1. This leads to enhanced, sharable MSC information semantics.[8]

Two cases are used in the analyses to show the canonicalization impacts. Case 1 is the OWL mapping between a canonicalized proprietary MSC model and the reference MSC ontology. Case 2 is the OWL mapping between a non-canonicalized proprietary MSC model and the reference MSC ontology. The non-canonicalized proprietary MSC model is an OWL-encoded proprietary MSC model using the RDB automatic conversion profile described in Section 4.3, i.e., it is the source ontology. As such, it does not follow the ODPs used by the reference MSC ontology. On the other hand, the canonicalized proprietary MSC model follows the same ODPs used by the reference MSC ontology. It is the result of performing

[8] Note: Our canonicalization process does not improve necessarily the other steps in that process.

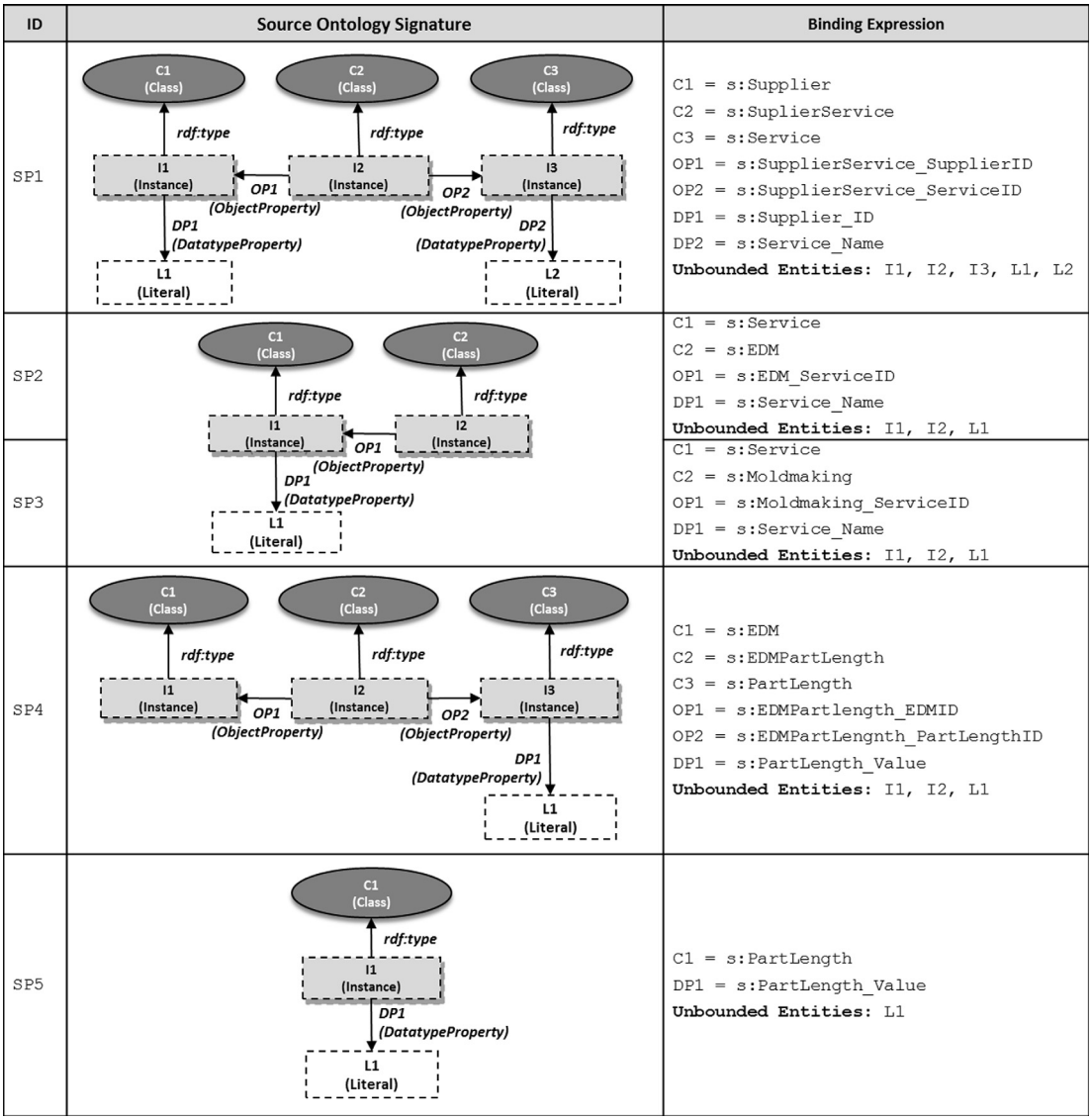


Fig. 15. Graphical representations of the source ontology patterns.

the transformations of the non-canonicalized schema and the corresponding data that are described in Section 5. The next two subsections provide a detailed discussion of these analyses.

6.1. Qualitative analysis

The qualitative analysis compares the OWL mapping statement complexity, mappability, and query behavior of the two cases. Queries, which use only terminologies from the reference MSC ontology, are used in the analysis. These queries retrieve information encoded in the proprietary terminologies and structures. Consequently, they verify the extent to which semantic mediation has occurred.

MSC information related to `Supplier_3` in the proprietary MSC data model shown in Fig. 11 is used for this illustration. The MSC information has two features: service category and part length capability. The queries `Q1` and `Q2` in Fig. 20 contain conditions for these two features; hence, they are sufficient to verify the semantic mediation related to these two features. These queries are encoded in OWL DL query language (W3C, 2009c) with the reference ontology terminology and structure. The expected results from both queries are `Supplier_3`.

First, we analyze Case 1: OWL mapping between a canonicalized proprietary MSC model and the reference MSC ontology. On the left side of Fig. 21 is the canonicalized proprietary MSC model of `Supplier_3`; and, on the right side is the reference MSC ontology. The dotted lines in the figure depict all the necessary mapping axioms between the two models via either the `owl:equivalentClass` or `owl:equivalentProperty` predicate. These axioms are sufficient for both `Q1` and `Q2` to return the correct results. Notice that both the subjects and objects of the mapping axioms are simply a single class or property name. There is no need for complex *OWL class or property expressions* (W3C, 2009b). This is because both models follow the same set of ODPs and are structurally aligned.

Next we analyze Case 2: OWL mapping between the original MSC source ontology, derived from its relational DB, and the reference MSC ontology. Fig. 22 illustrates this case where the left side shows `Supplier_3` information from the source ontology and on the right side shows the reference ontology. Unlike the previous case, simple class-to-class and property-to-property mapping axioms are insufficient to enable `Q1` and `Q2` to find the correct results. Specifically, there are two issues and they are highlighted in the shaded areas connected by the double arrow lines in the

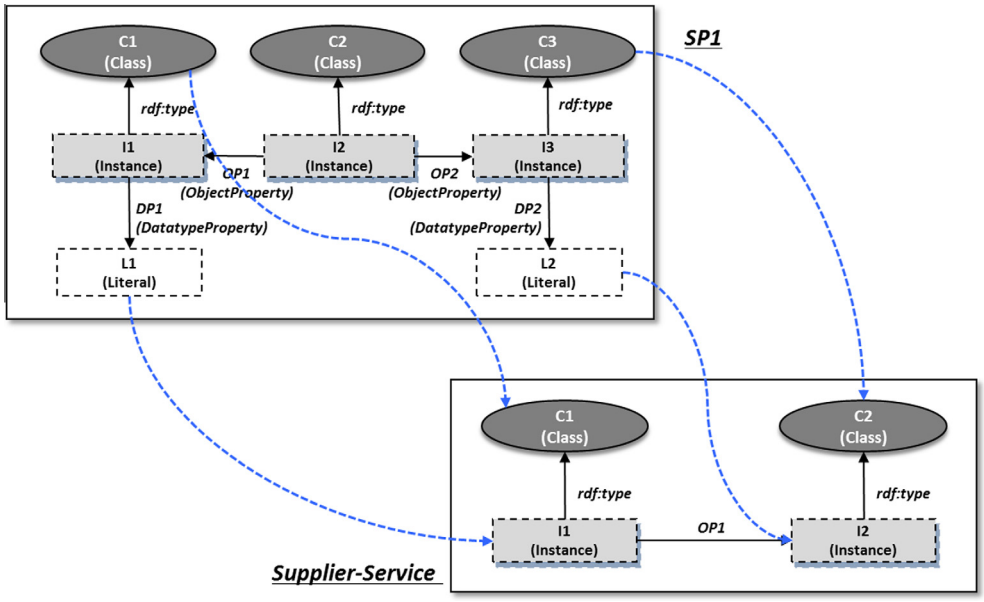


Fig. 16. PTRs between the source and target ontology patterns in OPC1.

Table 8
Pattern transformation rule set for OPC1.

PTRS ID	PTR ID	Source ontology pattern variable	Target ontology pattern variable	Transformation type	Transformation expression
PTRS1	PTR1.1	s:C1	p:C1	Type-1	s:C1
	PTR1.2	s:C3	p:C2	Type-1	s:C3
	PTR1.3	s:L1	P:I1	Type-2	s:L1
	PTR1.4	s:L2	P:I2	Type-2	s:L1
	PTR1.5	–	p:OP1	Type-4	p:hasService
	PTR1.6	s:I1	–	Type-3	–
	PTR1.7	s:I2	–	Type-3	–
	PTR1.8	s:I3	–	Type-3	–
	PTR1.9	s:OP1	–	Type-3	–
	PTR1.10	s:OP2	–	Type-3	–
	PTR1.11	s:DP1	–	Type-3	–
	PTR1.12	s:DP2	–	Type-3	–
	PTR1.13	s:C2	–	Type-3	–

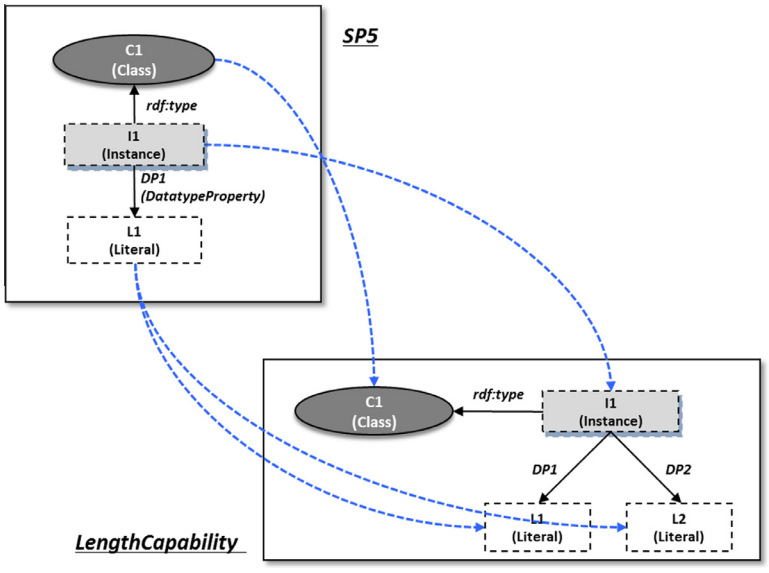


Fig. 17. Literal value pattern detection.

Table 9
Pattern transformation rule set for OPC5.

PTRS ID	PTR ID	Source ontology pattern	Target ontology pattern	Transformation type	Transformation expression
PTRS5	PTR5.1	s:C1	p:C1	Type-1	s:C1
	PTR5.2	s:I1	p:I1	Type-1	s:I1
	PTR5.3	s:L1	p:L1	Type-2	s:G1
	PTR5.4	s:L1	P:L2	Type-2	s:G2
	PTR5.5	–	p:DP1	Type-4	p:hasMin
	PTR5.6	–	p:DP2	Type-4	p:hasMax
	PTR5.7	s:DP1	–	Type-3	–

Table 10
OPCs ready for the pattern transformation.

OPC ID	Source ontology pattern	Target ontology pattern	Pattern transformation rule set
OPC1	SP1	Supplier-Service	PTRS1
OPC2	SP2	Service-Categorization	PTRS2
OPC3	SP3	Service-Categorization	PTRS3
OPC4	SP4	Service-LengthCapability	PTRS4
OPC5	SP5	LengthCapability	PTRS5

```
PREFIX rdf:  http://www.w3.org/1999/02/22-rdf-syntax-ns#
PREFIX s:  <http://www.nist.gov/el/sid/msnm/PortalB.owl#>

SELECT distinct *
    WHERE {
        ?I1 rdf:type s:Supplier .
        ?I2 rdf:type s:SupplierService .
        ?I3 rdf:type s:Service .
        ?I2 s:SupplierService_SupplierID ?I1 .
        ?I2 s:SupplierService_ServiceID ?I3 .
        ?I1 s:Supplier_ID ?L1 .
        ?I3 s:Service_Name ?L2 .
    }
```

Fig. 18. SPARQL query generated from SP1.

Table 11
Occurrences of SP1 in the source ontology derived from SP1 query.

SP1 instance ID	I1	I2	I3	L1	L2
SPI1.1	s:Supplier_3	s:SS_3_4	s:Service_4	''Supplier_3''	''S3_EDM_Service''
SPI1.2	s:Supplier_6	s:SS_6_9	s:Service_9	''Supplier_6''	''S6_Moldmaking_Service''

Table 12
Results of PTRS1 application on SPI1.1.

PTR ID	SP1 variables	Source ontology artifacts	Transformation type	Result	
				Name	Type
PTR1.1	s:C1	s:Supplier	Type-1	s:Supplier	Class
PTR1.2	s:C3	s:Service	Type-1	s:Service	Class
PTR1.3	s:L1	''Supplier_3''	Type-2	s:Supplier_3	Individual
PTR1.4	s:L2	''S3_EDM_Service''	Type-2	s:S3_EDM_Service	Individual
PTR1.5	–	–	Type-4	s:hasService	Object Property
PTR1.6	s:I1	s:Supplier_3	Type-3	–	–
PTR1.7	s:I2	s:SS_3_4	Type-3	–	–
PTR1.8	s:I3	s:Service_4	Type-3	–	–
PTR1.9	s:OP1	s:SupplierService_SupplierID	Type-3	–	–
PTR1.10	s:OP2	s:SupplierService_ServiceID	Type-3	–	–
PTR1.11	s:DP1	s:Supplier_ID	Type-3	–	–
PTR1.12	s:DP2	s:Service_Name	Type-3	–	–
PTR1.13	s:C2	s:SupplierService	Type-3	–	–

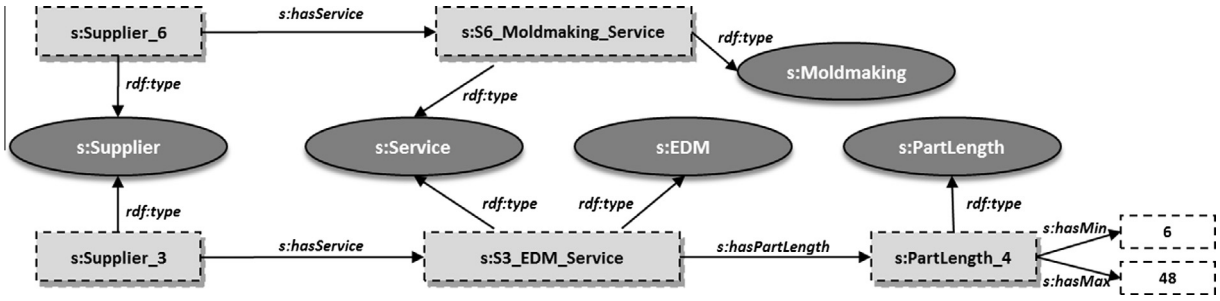


Fig. 19. Canonicalized proprietary MSC model.

```
Q1: Identify suppliers having the ElectroDischargeMachiningService which has minimum 6cm
and maximum 48cm as part length capability value using only terms from the standard
manufacturing service model.

Terms in the standard ontology are denoted by the prefix, 'mo'.

OWL DL Query => mo:Supplier and mo:hasService some (mo:ElectroDischargeMachiningService and
mo:hasLengthCapability some (mo:PartLengthCapability and ((mo:hasMin value 6^^xsd:double)
and (mo:hasMax value 48^^xsd:double))))
```

```
Q2: Identify suppliers having the ElectroDischargeMachiningService using only terms from
standard manufacturing service model.

Prefix: Terms in the standard ontology are denoted by the prefixes, 'mo'.
OWL DL Query => mo:Supplier and mo:hasService some mo:ElectroDischargeMachiningService
```

Fig. 20. Desired query behavior to demonstrate the OWL-based semantic mediation.

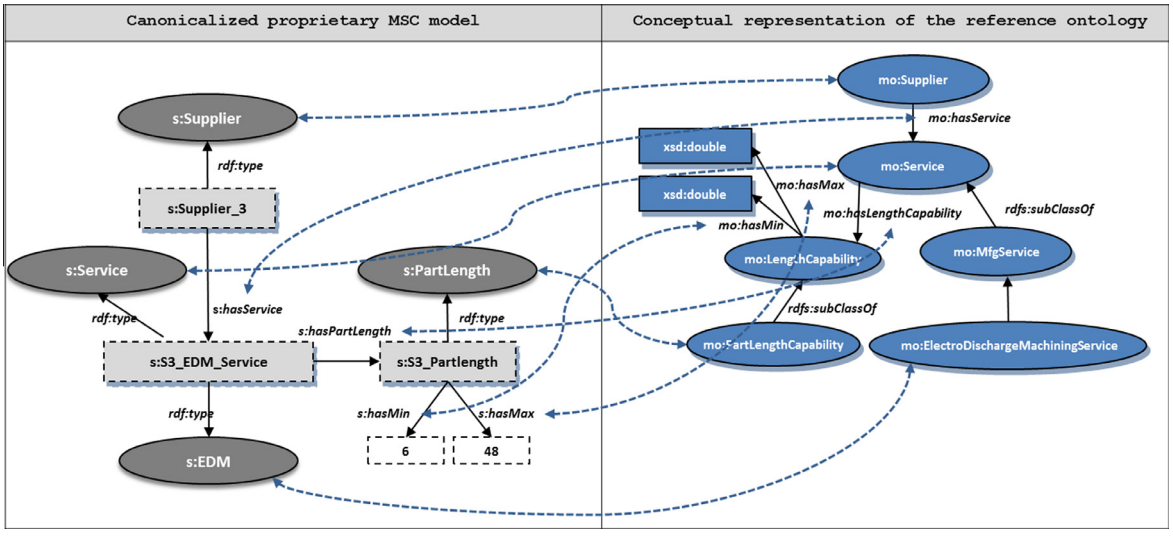


Fig. 21. Qualitative analysis of OWL mapping after canonicalization.

figure. First there is a need to map between differing OWL ontology artifact types, specifically from a literal, ''6cm – 48cm'', to two OWL data properties, `mo:hasMin` and `mo:hasMax`. This is related to the part length information and is needed to enable `Q1`; however, the required mapping cannot be constructed within OWL DL.

The other issue involves the creation of a bridge to deal with the different structural representations of the EDM manufacturing service in the source and reference ontologies. Such a bridge, which is necessary to enable `Q2`, can be created using the mapping class technique described in (Kulvatunyou et al., 2013). The required mapping class `EDMSupplier` is shown in Table 13. A mapping class is a defined-class with multiple necessary and sufficient conditions (`owl:equivalentClass` axioms). Each condition uses terms from a single terminology set and requires an OWL class expression as an object in the axiom. In this case, the two mapping class axioms, `A1` and `A2`, are shown in Table 13. Although these mapping class axioms enable `Q2` to return the desired results, the query behavior associated with the mapping class technique is somewhat restricted. The reason is that the bridge between the views is limited to the class expressions provided in the axioms.

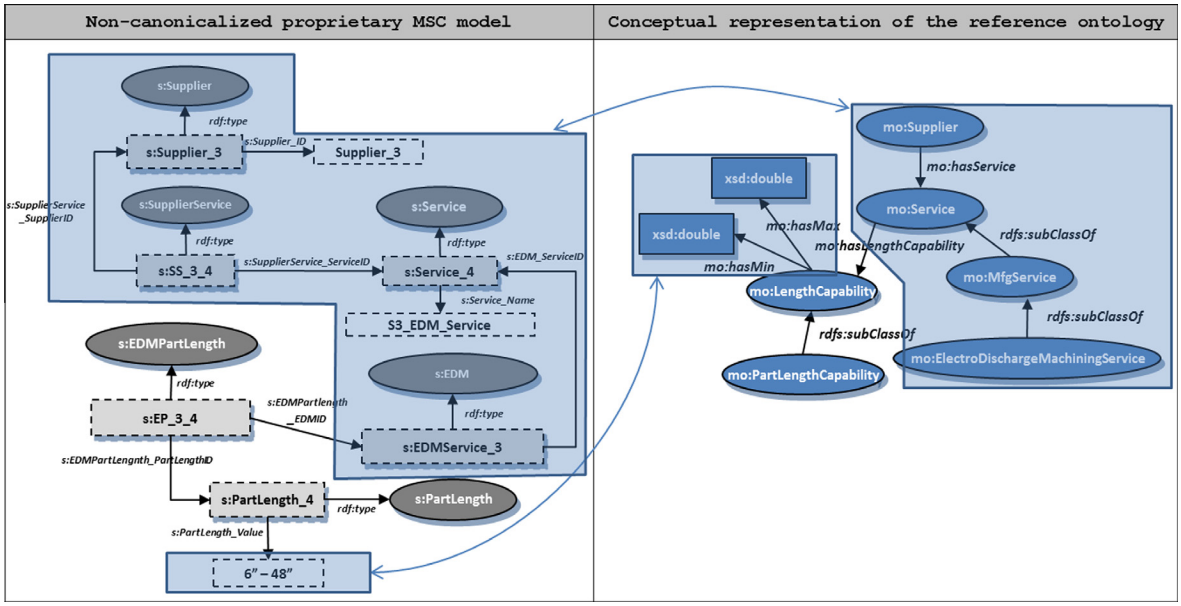


Fig. 22. Qualitative analysis of OWL mapping without canonicalization.

Table 13
Mapping class axioms.

Mapping class	Axiom ID	Mapping class axioms
EDMSupplier	A1	mo:Supplier and mo:hasService some mo:ElectroDischargeMachiningService
	A2	s:Supplier and inverse s:SupplierService_SupplierID some (s:SupplierService and s:SupplierService_ServiceID some (s:Service and inverse s:EDM_ServiceID some s:EDM))

For example, the EDMSupplier mapping class in A1 is related only to mo:Supplier and mo:hasService. If a new target of the query is related to, say, mo:Factory and mo:hasService, another mapping class will be needed. Moreover, the number of mapping classes grows proportionally with the number of service categories.

In summary then, this qualitative analysis shows that our canonicalization approach can circumvent OWL DL limitations and simplify the mappings. Additionally, it can simplify the mapping maintenance by reducing the number and complexity of mapping axioms.

6.2. Quantitative analysis

The quantitative analysis focuses on OWL inference times. Specifically, we study the impact of the number of mapping classes on the amount of time required to complete the OWL DL reasoning process. The reasoning process consists of four tasks: loading, consistency checking, classification, and realization as described in (MINDSWAP, 2012). All the reasoning is performed on a desktop running Pellet version 2.2.2 (Clark and Parsia, 2012).

In this analysis, we use only the MSC information related to the supplier's service declaration in both the source and the reference ontologies. We show that even if a human applies additional transformation rules to the proprietary model without considering the design pattern used in the reference ontology, the two ontologies may still be structurally misaligned and increase reasoning inefficiency, and therefore, reasoning time. Fig. 23 shows this additional lift-up based on the proprietary semantic interpretation. A single relationship via the s:provideService object property and the s:EDM instance of the s:Service class replaces the whole complex structure to declare the EDM (Electro-discharge Machining) service capability.

To analyze the effect on reasoning time, this modeling pattern, which represents service category as an instance of the service class, is replicated with polymorphic names (C1, C2, ..., Cn) for several declarations of service capabilities. The reference ontology, on the other hand, represents each service category as a subclass of the service class. Different sets of mapping axioms are then required in the case of canonicalized and non-canonicalized proprietary MSC model as shown in the middle of Figs. 24 and 25, respectively.

Since the canonicalized proprietary MSC model is structurally aligned with the reference MSC ontology, the service categories s:C1 to s:Cn are modeled the same way with OWL classes, mo:C1 to mo:Cn. As a result, the mapping consists of only simple class-to-class and property-to-property equivalence mapping axioms as shown in the figure. On the other hand, different modeling patterns between the non-canonicalized proprietary MSC model and the reference ontology necessitate the mapping classes. Differing mapping classes are required for C1 to Cn. In general, an additional mapping class is needed for each additional service category. For the quantitative analysis, we perform 10 semantic mediation experiments and compare reasoning times between the two cases by incrementing the number of service categories by 10 for each experiment up to 100 (n = 100).

Table 14 shows the reasoning times in the canonicalization case. It shows that increases in the number of service categories has (1) little impact on the consistency checking and classification times and (2) minor increases on the loading and realization times. Table 15 shows the reasoning times in the non-canonicalization. In this case, the classification and realization times are significantly affected by the increase in the number of service categories; while the loading and consistency-checking times are marginally increased. The graph in Fig. 26 concludes that the total reasoning time in the non-canonicalization case grows cubically with the number of service category classes versus linearly in the canonicalization case. This finding suggests that canonicalization can play a

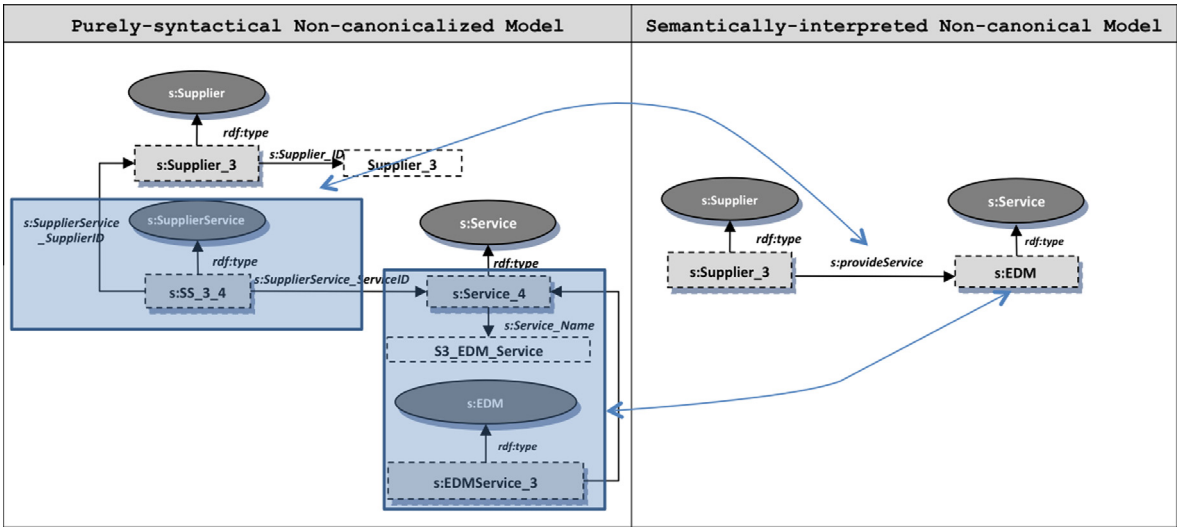


Fig. 23. Semantically-interpreted non-canonical model.

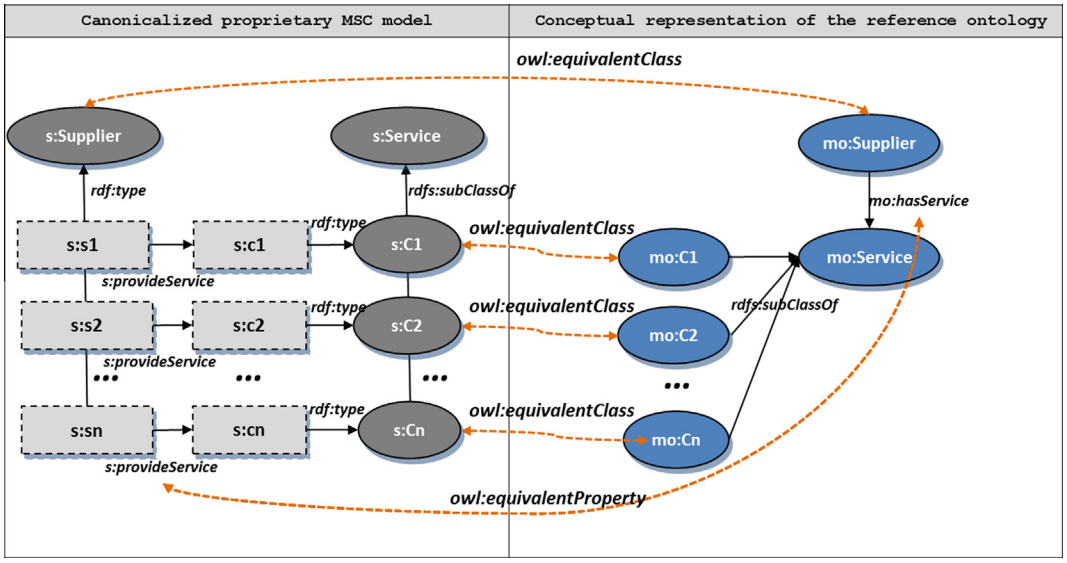


Fig. 24. Quantitative analysis of OWL mapping inference after canonicalization.

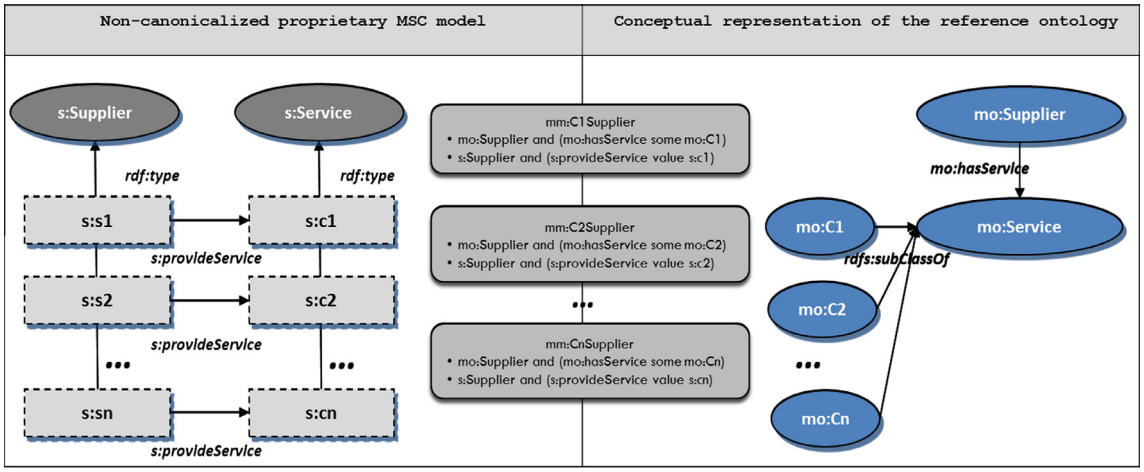


Fig. 25. Quantitative analysis of OWL mapping inference without canonicalization.

Table 14
Reasoning times in the canonicalization case (all times are in second).

Number of service category classes	Loading time	Consistency checking time	Classification time	Realization time	Total time
10	0.005	0.001	0.001	0.001	0.008
20	0.006	0.001	0.001	0.002	0.010
30	0.009	0.002	0.001	0.003	0.015
40	0.01	0.001	0.001	0.007	0.019
50	0.012	0.002	0.001	0.009	0.024
60	0.013	0.003	0.001	0.012	0.029
70	0.016	0.002	0.001	0.017	0.036
80	0.02	0.005	0.001	0.023	0.049
90	0.022	0.004	0.001	0.026	0.053
100	0.024	0.005	0.002	0.031	0.062

Table 15
Reasoning times in the non-canonicalization case (all times are in second).

Number of service category classes	Loading time	Consistency checking time	Classification time	Realization time	Total time
10	0.031	0.031	0.203	0.062	0.327
20	0.047	0.032	1.67	0.391	2.14
30	0.062	0.047	10.744	2.17	13.023
40	0.063	0.156	38.508	6.184	44.911
50	0.047	0.25	91.101	13.039	104.437
60	0.062	0.343	185.847	24.81	211.062
70	0.062	0.421	339.18	44.652	384.315
80	0.078	0.687	573.335	78.936	653.036
90	0.078	0.796	948.925	124.508	1074.307
100	0.078	1.046	1505.469	183.522	1690.115

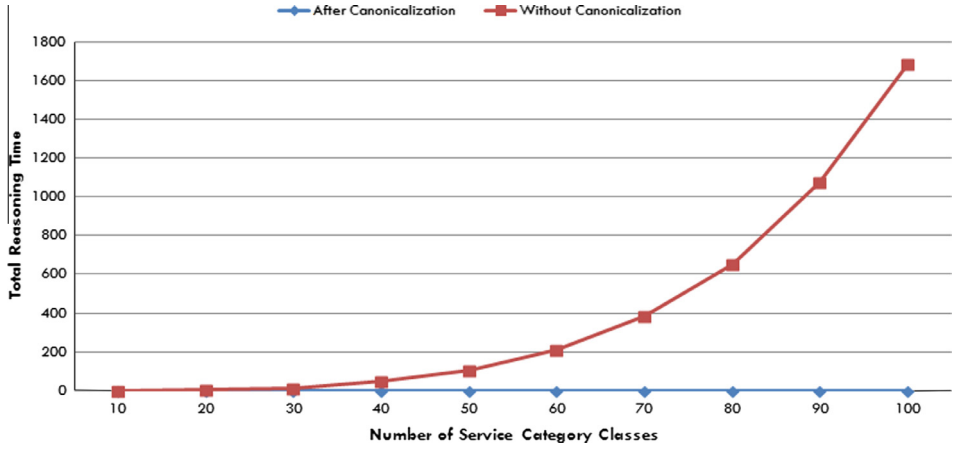


Fig. 26. Aggregated reasoning performances of two cases.

significant role in a practical deployment of OWL DL-based semantic mediation when there are structural conflicts.

7. Conclusion and future works

This paper describes a novel approach to semantic mediation by decomposing the mapping task into two steps. The first step resolves the structural conflicts between the source schemas and the reference ontology using a "canonicalization" transformation. The second step addresses other conflicts using OWL DL mapping axioms. The paper formalized a canonicalization framework and demonstrated its applicability using a realistic example of a proprietary manufacturing service capability database.

The primary contribution of this framework is the synthesis of complementary pieces of work in syntactical data transformation, ontology design patterns, ontology matching, and pattern-based ontology transformation. In addition, the framework outlines a novel approach to representing reusable, conceptual, ontology design patterns and capturing the ontology pattern correspondences (OPCs) in the source ontology. The proposed framework was evaluated using both a qualitative and a quantitative analysis. The qualitative analysis shows that canonicalization can circumvent OWL DL limitations to increase the mappability and can simplify the OWL mapping axiom. The quantitative analysis shows that OWL reasoning time grows (1) cubically when OWL DL axioms are used to resolve a common type of structural conflict but (2) linearly using our canonicalization approach.

In terms of future work, we plan to develop methods and tools to implement the proposed framework. This includes (1) pattern library storage, which supports the proposed conceptual ontology design pattern representation, (2) a computer-assisted OPC identification environment that employs ontology matching algorithms, (3) a software component that helps users manage the OPC identification process, (4) enhancements to the existing pattern transformation engine, and (5) a methodology for deriving ontology design patterns for the manufacturing service capability information.

Disclaimer

Certain commercial software products are identified in this paper. These products were used only for demonstration purposes. This use does not imply approval or endorsement by NIST, nor does it imply these products are necessarily the best available for the purpose.

Acknowledgement

The work described in this paper was funded in part by NIST through University of Maryland, Baltimore County cooperation agreement #70NANB13H154.

References

Ameri, F., & Dutta, D. (2006). An upper ontology for manufacturing service description. In *ASME 2006 international design engineering technical conferences and computers and information in engineering conference* (pp. 651–661). American Society of Mechanical Engineers.

Bizer, C. (2003), D2R MAP: A database to RDF mapping language. In *Proceedings of the 12th international world wide web conference, Budapest, Hungary, 2003*.

Bizer, C., & Seaborne, A. (2004), D2RQ–treating non-RDF databases as virtual RDF graphs. In S.A. Mcllraith, D. Plexousakis, F. van Harmelen (Eds.), *Proceedings of 3rd international semantic web conference (Hiroshima, Japan, 2004)* (Vo. 3298), Springer, Lecture Note in Computer Sciences.

Bloomfield, R., Mazhari, E., Hawkins, J., & Son, Y. J. (2012). Interoperability of manufacturing applications using the Core Manufacturing Simulation Data (CMSD) standard information model. *Computers & Industrial Engineering, 62*(4), 1065–1079.

Clark and Parsia LLC (2012). *Pellet: OWL 2 reasoner for Java version 2.2.2*. <http://clarkparsia.com/pellet/> Accessed May 2014.

D2RQ framework version 0.8.1. *Accessing relational databases as virtual RDF graphs*. <http://d2rq.org/> Accessed May 2014.

Das, S., Sundara, S., & Cyganiak (2012). *R. R2RML: RDB to RDF mapping language*. World Wide Web Consortium, Recommendation REC-r2rml-20120927, September 2012.

Gangemi, A. (2005). Ontology design patterns for semantic web content. In *International semantic web conference. Lecture Note in Computer Sciences* (Vol. 3729, pp. 262–276). Berlin: Springer-Verlag.

Kulvatunyou, B., Ivezic, N., Lee, Y., & Shin, J. (2013). An analysis of OWL-based semantic mediation approaches to enhance manufacturing service capability models. *International Journal of Computer Integrated Manufacturing*, 1–21 (ahead-of-print).

Lu, Y., Panetto, H., Ni, Y., & Gu, X. (2013). Ontology alignment for networked enterprise information system interoperability in supply chain environment. *International Journal of Computer Integrated Manufacturing, 26*(1–2), 140–151.

McGuinness, D. L., Fikes, R., Rice, J., & Wilder, S. (2000). The chimaera ontology environment. In: *AAAI/IAAI 2000* (pp. 1123–1124).

Min, H., & Zhou, G. (2002). Supply chain modeling: Past, present and future. *Computers & Industrial Engineering, 43*(1), 231–249.

MINDSWAP – Maryland Information and Network Dynamics Lab Semantic Web Agents Project (2012), Pellet Performance Report. <http://www.mindswap.org/2003/pellet/performance.shtml> Accessed September 2012.

Noy, N. F., & Musen, M. A. (2003). The PROMPT suite: Interactive tools for ontology merging and mapping. *International Journal of Human-Computer Studies, 59*(6), 983–1024.

OPPL – Ontology Pre-Processor Language version 2 (2012). <http://oppl2.sourceforge.net/> Accessed September 2012.

Park, J., & Ram, S. (2004). Information systems interoperability: What lies beneath? *ACM Transactions on Information Systems, 22*(4), 595–632.

Presutti, V., Gangemi, A., David, S., Aguado de Cea, G., Suárez-Figueroa, M. C., Montiel-Ponsoda, E., et al. (2008), *NeOn project delivery – D2.5.1. A library of ontology design patterns: Reusable solutions for collaborative design of networked ontologies*.

Satya, S. S., Halb, W., Hellmann, S., Idehen, K., Thibodeau, T., Auer, S., et al. (2009), *A survey of current approaches for mapping of relational databases to RDF, W3C RDB2RDF incubator group*. <http://www.w3.org/2005/Incubator/rdb2rdf/RDB2RDF_SurveyReport.pdf>.

Sheth, A. P., & Kashyap, V. (1992). So far (schematically), yet so near (semantically). In D. K., Hsiao, E. J. Neuhold, R. Sacks-Davis (Eds.), *Proceedings of the IFIP WG2.6 database semantics conference on interoperable database systems* (DS-5, Lorne, Victoria, Australia, November 16–20) (pp. 283–312).

Shvaiko, P., & Euzenat, J. (2011). Ontology matching: State of the art and future challenges. *IEEE Transactions on Knowledge and Data Engineering*, 99.

SMLC – Smart Manufacturing Leadership Coalition (2011). *Implementing 21st century smart manufacturing*, Workshop Summary Report, June 24, 2011. <https://smart-process-manufacturing.ucla.edu/about/news/Smart%20Manufacturing%206_24_11.pdf>.

Svab-Zamazal, O., & Svatek, V. (2011), OWL matching patterns backed by naming and ontology patterns. In *Znalosti, 10th Czecho-Slovak knowledge technology conference*, Stara Lesna, Slovakia.

Svab-Zamazal, O., Svatek, V., Scharffe, F., & David, J. (2009), Detection and Transformation of Ontology Patterns, Knowledge Discovery, Knowledge Engineering and Knowledge Management, Revised Selected Papers from IC3K. Springer CCIS no.128, 2011, 210–223.

Tsinaraki, C., Polydoros, P., & Christodoulakis, S. (2004). Interoperability support for ontology-based video retrieval applications. *Image and Video Retrieval, 3115*, 582–591.

W3C – World Wide Web Consortium (2004a). SWRL: A Semantic Web Rule Language, May 21, 2004. <http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/>.

W3C – World Wide Web Consortium (2004b). Resource Description Framework (RDF): Concepts and Abstract Syntax, February 10, 2004. <http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/>.

W3C – World Wide Web Consortium (2004c), XML Schema, Parts 0, 1, and 2 (2nd ed.), October 28, 2004. <http://www.w3.org/TR/xmlschema-0/, http://www.w3.org/TR/xmlschema-1/, and http://www.w3.org/TR/xmlschema-2/>.

W3C – World Wide Web Consortium (2005a). Representing Classes As Property Values on the Semantic Web, April 5, 2005. <http://www.w3.org/TR/swbp-classes-as-values/>.

W3C – World Wide Web Consortium (2005b). Representing Specified Values in OWL: "value partitions" and "value sets", May 17 2005. <http://www.w3.org/TR/swbp-specified-values/>.

W3C – World Wide Web Consortium (2006). Extensible Markup Language (XML) 1.1, August 16, 2006. <http://www.w3.org/TR/xml11/>.

W3C – World Wide Web Consortium (2008). SPARQL Query Language for RDF, January 15, 2008. <http://www.w3.org/TR/rdf-sparql-query/>.

W3C – World Wide Web Consortium (2009a). OWL 2 Web Ontology Language, October 27, 2009. <http://www.w3.org/TR/owl2-overview/>.

W3C – World Wide Web Consortium (2009b). OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax, October 27, 2009. <http://www.w3.org/TR/owl2-syntax/>.

W3C – World Wide Web Consortium (2009c). OWL 2 Web Ontology Language Manchester Syntax W3C Working Group Note, October 27, 2009. <http://www.w3.org/TR/owl2-manchester-syntax/>.

Wang, G., Wong, T. N., & Wang, X. (2013). An ontology based approach to organize multi-agent assisted supply chain negotiations. *Computers & Industrial Engineering*. http://dx.doi.org/10.1016/j.cie.2012.06.018.

Ye, Y., Yang, D., Jiang, Z., & Tong, L. (2007). An ontology-based architecture for implementing semantic integration of supply-chain management. *International Journal of Computer Integrated Manufacturing, 21*(1), 1–18.

Zheng, L., & Terpenny, J. (2013). A hybrid ontology approach for integration of obsolescence information. *Computers & Industrial Engineering*. http://dx.doi.org/10.1016/j.cie.2013.02.011.