A Framework to Canonicalize Manufacturing Service Capability Models

Boonserm Kulvatunyou[1], Yunsu Lee[1, 2], Nenad Ivezic[1], Yun Peng[2]

[1]Systems Integration Division, National Institute of Standards and Technology
Gaithersburg, MD 20899, U.S.A.

[2]Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County
Baltimore, MD 21250, U.S.A.

Abstract

The ability to share precise models of suppliers' *manufacturing service capability* (MSC) information is necessary to develop reliable methods that enable OEMs to efficiently configure agile and responsive supply chains. Currently, most suppliers use online tools to represent and share their MSC information in proprietary ways via *proprietary MSC data models*. These models have limited precision and interoperability. A semantically precise and rich reference MSC ontology can address both of these limitations and enable development of the reliable supply chain configuration methods. To effectively develop and deploy such a reference MSC ontology, semantic mediation between proprietary MSC models and the reference MSC ontology will be required. An important and challenging activity within the semantic mediation process is the mapping between a proprietary MSC data model and the reference MSC ontology. The challenge of the mapping activity is to resolve structural and semantic conflicts between the proprietary model and the reference ontology in a manner that is efficient and results in mapping structures that are simple to comprehend and maintain. This paper proposes an approach to address the challenge by preprocessing the structural representations of proprietary MSC data models for alignment with the set of modeling conventions (i.e., ontology design patterns - ODPs) that are also used in the reference MSC ontology. We call this preprocessing canonicalization. Canonicalization can circumvent 1:n, n:1, or n:m mapping statements that require complex expressions thereby simplifying the mapping activity and its resulting mapping statements. The main contribution of this paper is the design and formalization of an ODP-based canonicalization framework and its associated process in the context of description logic-based semantic mediation using the Ontology Web Language (OWL).

1 Introduction

The Smart Manufacturing Leadership Coalition has concluded that manufacturing service capability (MSC) information must be shared effectively across companies, to enable agile and optimized supply chains (SMLC 2011). That is, *MSC information* must be semantically precise, sufficient, and accessible in an interoperable way (Ameri and Dutta 2006). Currently, manufacturing companies share MSC

information through proprietary information models called MSC data models. Examples of these *proprietary MSC data models* are those found in commercial supplier information sharing portals including Mfg.com[1], GlobalSpec[2], and ThomasNet[3].These proprietary MSC data models are heterogeneous in their structures and data representations. In this situation, manufacturers cannot effectively share MSC information. Researches have shown that use of the Web Ontology Language (OWL) (W3C 2009a) reference ontology enhances access and precision of information). In particular, Ye et al. (2007), Lu et al. (2013), and Zheng and Terpenny (2013) use Semantic Web Rule Language (SWRL) (W3C 2004a) to link local and reference ontology, while Kulvatunyou et al. (2013) and Tsinaraki et al. (2007) use only OWL axioms.

The *OWL-based semantic mediation* in Kulvatunyou et al. (2013) uses OWL reasoner and OWL mapping axioms to inherit semantics from a semantically rich *reference MSC ontology*[4] resulting in enhanced semantic precision and coverage as well as semantic conflicts resolutions across proprietary MSC data models. The OWL-based semantic mediation can enable effective sharing of MSC information because the reference MSC ontology provides a common domain model and terminology that facilitates queries and responses across proprietary data models. The process of mediation described in Kulvatunyou et al. (2013) is as follows. MSC information in the proprietary MSC data models is first transformed into the common RDF syntax (W3C 2004b) using the OWL semantics resulting in *OWL-encoded proprietary MSC model*, which is then mapped to the reference ontology using OWL axioms. The description logic inference over OWL-encoded proprietary MSC models, reference ontology, and mapping axioms results in improved MSC information sharing.

In fact, the first step in semantic mediation of any information in OWL, including MSC information, is to convert the proprietary information into OWL. There are typically two ways to transform proprietary MSC data models (or any other kinds of data models) into OWL—purely syntactical or with semantic interpretation. In the first case, generic transformation rules based on the underlying schema language are applied to the data source. For example, in the case of relational databases, tables are transformed into classes and columns are transformed into properties. In the other case, rules specific to data source scheme are written by human and used for the transformation. This transformation may occur in a single hop or two hops (in which case post-processing based on the data-source-specific rules occurs after a syntactical transformation). The data-source-specific rules are typically derived from the view of the data source owner without regard to other views and requirements of similar MSC information. The key is that, in either case, the resulting models, called *arbitrary OWL-encoded proprietary MSC model*[5], are typically not structurally aligned with the target reference ontology as illustrated on the top of

[1] http://www.mfg.com
[2] http://www.globalspec.com
[3] http://www.thomasnet.com
[4] 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 millimeters 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.
[5] 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 proprietary view of the data.

Figure 1. Such arbitrary OWL-encoded proprietary MSC model can render the OWL mapping axioms exceedingly complex, if at all possible, in the OWL-based semantic mediation. Technologies such as the D2RQ (D2RQ 2012) and the W3C's R2RML (Das et al. 2012) support both the pure syntactical as well as the data-source-specific transformation practices.

Figure 1 should be inserted about here.

In this paper, we propose a methodology, called canonicalization, to streamline the OWL-based semantic mediation process in order to simplify the OWL mapping axioms and circumvent OWL axioms and reasoner limitations to produce mapping, particularly when dealing with structural differences. The idea is that a proprietary MSC data model is first automatically transformed with a syntactical standard rule set independent of its source data scheme; and then, with a human assistance, the canonicalization transforms the data again using a set of design patterns before writing OWL mapping axioms against the reference ontology. Since the design patterns used in the canonicalization are also used for the reference ontology, the resulting *canonicalized OWL-encoded proprietary MSC model* is more structurally aligned and simpler to map with the reference ontology. This proposed methodology is illustrated at the bottom of Figure 1.

The simplification of the OWL mapping axioms will be validated on an example; and quantitative and qualitative analyses of the simplification results will be provided. The qualitative analysis will show that canonicalization can amend a model not originally suited for semantic mediation via OWL DL, simplify the mapping by avoiding the need for complex OWL class expressions in the mapping axioms, and simplify the mapping maintenance by reducing the number 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 characterize canonicalization by the types of semantic conflicts it can address. Section 3 introduces the proposed canonicalization framework. It is followed with Section 4, which validates the applicability and usefulness of the framework with a running example and the qualitative and quantitative analyses. Finally, we provide insights into related works that can enable and improve the framework upon its deployment before giving a conclusion and remarks on the current work and our future plans.

2 Canonicalization Defined

In the context of semantic mediation, canonicalization may typically be viewed as preprocessing otherwise diverse proprietary representations to simplify mappings. In this paper, we follow logical and conceptual ontology design patterns (explained later in the section) in transforming the OWL-encoded proprietary manufacturing service capability (MSC) models in order to simplify the mappings to the reference MSC ontology.

Canonicalization in this paper is a means to avoid certain conflicts in encodings of proprietary and reference MSC information models. Sheth and Kashyab (1992) have identified various types of

schematic differences between semantically similar objects from the relational databases perspective (i.e., objects are tables). Park and Ram (2004) have characterized these differences into two broad categories, namely the *data-level* and *schema-level* conflicts. Most of these conflict types can be carried over to OWL-based models.

Data-level conflicts are differences in data domains caused by the multiple representations and interpretations of similar data. Data-level conflicts are applicable to representation of values of OWL data properties. Types of data-level conflicts relevant to our work include *data-representation* conflicts, *data-unit* conflicts, and *data-precision* conflicts. Data-representation conflicts occur when the semantically same values are represented differently such as `05/08/2012` and `May-08-2012`. The data-unit conflicts occur when the same quantities are represented with differing units, e.g., "2 inches" and "5 centimeters". Data-precision conflicts occur when different scaling is used, e.g., when continuous numerical numbers between 0 and 100 are used to indicate qualities vs. when discrete scale like low, medium, high is used.

The schema-level conflicts are subcategorized into *naming conflicts*, *entity-identifier conflicts*, *schema-isomorphism conflicts*, *generalization conflicts*, *aggregation conflicts*, and *schematic discrepancies*. Naming conflicts are the cases where 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 as they have names. 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 refer to the same individual. Isomorphism conflicts are the cases where two semantically same concepts are modeled with differing set of attributes and also differing number of attributes, e.g., `Supplier(ID, GeneralPhone, SupportPhone)` and `Supplier(ID, Phone)`, `Address(Line1, Line2, Zip)` and `Address(Street, City, State, Zip)`. Isomorphism conflicts are applicable to OWL classes in the sense that they can have differing set of properties. Generalization conflicts are the cases where objects/classes subsume one another, e.g., `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 are the cases 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)`. Aggregation conflicts are applicable to OWL classes. The schematic discrepancies are the cases where information is modeled using differing constructs – table name, attribute name, and attribute value. In OWL, the information about a supplier providing a CNC Machining Service may be modeled using a class declaration axiom (a supplier is a type of `CNCMachiningService` class), an object property assertion (e.g., the supplier has an object property pointing to an instance of `CNCMachiningService` class or the supplier has an object property pointing to a `CNCMachiningService` instance of a `ManufacturingService` class), or a data property assertion (e.g., the supplier has a string-based property `providesService` pointing to "`CNCMachiningService`", the supplier has a boolean property `isCNCMachiningServiceProvider` with the value true).

In our work, canonicalization is rooted in encoding the proprietary MSC data model such that it follows a set of design patterns. Gangemi (2005) has described two types of ontology design patterns, logical and conceptual. Logical ontology design patterns are independent of conceptualization, while conceptual ontology design patterns are specific to a domain of classes and properties. Examples of logical ontology design patterns are those given by the W3C Semantic Web Task Force on Ontology Engineering Patterns including `Representing Classes As Property Values on the Semantic Web`, `Representing Specified Values in OWL: "value partitions" and "value sets"`, and others (W3C 2005a, W3C 2005b). An example of the conceptual ontology design pattern given in Gangemi (2005) is `Participation at spatio-temporal location`. An example of the conceptual ontology design pattern related to MSC information is `Dimensional Capability Expression`. The pattern recommends that dimensional capability be represented as a class with two data properties representing minimum and maximum values. A proprietary manufacturing service capability data model may have its part envelope size capability represented as a text value such as `"5 – 25 centimeters"`. Canonicalization would convert such value into the representation recommended by the `Dimensional Capability Expression` pattern.

Canonicalization in this paper focuses on resolving particular schema-level conflicts described above, particularly the isomorphism conflicts and schematic discrepancies. Resolution of the data-level conflicts may be incorporated into the canonicalization; however, it is out of scope of the paper.

To be more specific, we further subcategorize the isomorphism conflicts into two cases, *isomorphism conflicts with the same data precision* and *isomorphism conflicts with differing data precision*. Canonicalization helps avoid only the former case. The latter case may be better left to handle via the OWL's description logic based mapping.

The isomorphism conflicts with the same data precision are the cases where two objects contain the same set of information. We extend this to also include the cases where data precision can be made the same by deriving the information from within the model or external to the model. For example, the `MaximumPartSize(value)` of an EDM machine can be canonicalized according to the `DimensionalCapability(minValue, maxValue)` ontology design pattern to become `PartSizeCapability(minValue, maxValue)` where the `maxValue = value` and the `minValue` can either be defaulted to zero or be derived from common domain knowledge.

The isomorphism conflicts with differing data precision are the cases where two objects have differing sets of information and that the differences cannot be derived or defaulted. For example, the `EDMService(ID, Type, SpecialService)` and `EDMMachiningService(ID, Type, SpecialService, MaxPartLength)` objects have differing set of information. Canonicalization does not resolve this type of conflict. OWL DL mapping axioms can be used to merge the two set of properties given that the two objects/concepts are equivalent.

The isomorphism conflicts and the schematic discrepancies may interact to form complex structural conflicts. Therefore, broadly speaking in this paper canonicalization aims at resolving the structural conflicts where the multitude of logical structures of a set of properties, their values, and axioms belonging to a class and/or instances in one model (a proprietary MSC model) are re-organized to form a

different structure in a canonical model (a reference ontology) which follows a set of consistent design patterns.

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).

3 Canonicalization Framework

This section describes the proposed canonicalization framework. The first subsection describes terminology related to OWL that will be used throughout the rest of the discussion.

3.1 Relevant OWL Terminologies

According to OWL 2 Structural Specification (W3C 2009b), there are three types of atomic symbols including *entities*, *literals*, and *anonymous individuals*. Entities are fundamental building blocks of an OWL ontology. They define named terms of an ontology and are uniquely identified by IRIs (Internationalized Resource Identifiers, an identifier construct akin to Uniform Resource Identifiers – URIs) (RFC3987). Classes, datatypes, object properties, data properties, annotation properties, and named individuals are entities. Literals are data values such as strings or integers. They have an associated datatype specifying how to interpret the value (lexical form). Anonymous individuals are akin to the named individuals but with a system assigned IRI. These are blank nodes in RDF (W3C 2009b).

The main component of an OWL ontology is a set of *axioms*. Axioms are statements that say what is true in the domain. Entities, literals, and anonymous individuals are used to compose an axiom. Axioms in OWL ontology can be declaration axioms, class axioms, object property axioms, data property axioms, datatype definitions, keys, assertions (sometimes also called facts), and annotation axioms. For the purpose of our discussion, entities, literals, and axioms are collectively referred to as *ontology artifacts*.

The rest of section 3 delves into the detail of the proposed canonicalization framework.

3.2 Overview

This section describes the proposed canonicalization framework, which is outlined in Figure 2. The initial input to the canonicalization is a proprietary MSC data model. Differing proprietary MSC data models may use differing 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. The step should be generally automatic using a standard transformation rules 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*[6] which is an input into the following OPCs (ontology pattern correspondence) identification step. Another input to the OPCs identification step is the *pattern library*. A pattern library consists of conceptual *OWL ontology design patterns* (ODPs for short) each of which indicates suitable structural pattern for an archetypical

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

unit of manufacturing service relations encoded in OWL DL. In this OPCs identification step, correspondences between the fragments of source ontology and ODPs are specified (from here on, ODPs refer specifically to OWL ODPs unless otherwise stated). This output from the OPCs identification is called *ontology pattern correspondences* (OPCs). The OPCs are used to construct *source ontology patterns* in the *source ontology patterns generation* step and also to retrieve the applicable *target ontology pattern* from the pattern library. A source ontology pattern is used to retrieve from the source ontology all matching *pattern instances*. A pattern instance is a source ontology fragment to be transformed with respect to a target ontology pattern. How these artifacts are transformed is represented by the *transformation rules* that are generated in the *transformation rules generation* step. The final pattern transformation step applies the source ontology patterns and executes the respective transformation rules (by using a target ontology pattern) 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.

Figure 2 should be inserted about here.

3.3 OWL Ontology Design Pattern

In this paper, an ODP is a reusable successful solution to a recurrent semantic modeling problem, written in OWL (Gangemi 2005). ODPs can be viewed as generic, small ontologies or ontology components with explicit documentation of design rationales and best reengineering practices. Pattern-based approach for ontology design has been gaining popularity recently because by reusing existing tested patterns as building blocks, a domain ontology can be constructed quickly with high quality and less conceptualization divergence. A large amount of ODPs (OWL ODPs and other language independent patterns) have been proposed in the ontology design community (Presutti et al. 2008)[7]. In this paper, we define a formal representation of 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 $\{SL, SX\}$

- 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
- $C \subset E \cup L$, 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[8] is represented by an ODP signature and a set of binding expressions to a set of entities and literals from the archetypical ontology. An ODP signature is a parameterized structure of an archetypical ontology, while the binding expressions connect parameters in the ODP signature to the archetypical entities and literals in the archetypical ontology fragment. These entities and literals are concepts and values giving a specific meaning to the ODP signature. The entities and literals within the archetypical ontology fragment are divided into two groups 1) conceptual and 2) representative; and only the conceptual entities and literals are used in the binding expressions to convey the meaning of the ODP.

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

[8] 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 ODP's 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).

The representative entities and literals represent the varying part of the pattern that need to be replaced by entities and literals from the source ontology or by defaulted values. Figure 3 and Figure 4 illustrate this through a `Supplier-Service` ODP example. Figure 3 shows the archetypical ontology fragment (conceptual pattern) of the ODP, while Figure 4 shows the ODP represented by a signature and binding expressions. With respect to the archetypical ontology fragment in Figure 3 and ODP in Figure 3, `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).

Figure 3 should be inserted about here.

Figure 4 should be inserted about here.

Table 1 should be inserted about here.

3.4 Transformation

The Transformation step applies the standard rule-based transformation to commonalize heterogeneous syntaxes of proprietary MSC data models into the RDF graph syntax using OWL DL vocabulary and semantics. The output of this step is called source ontology which corresponds to the intermediate OWL-encoded proprietary MSC model in Figure 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). Tools to support such automation are discussed in the related work section. In our work, the proprietary MSC data models are captured in relational databases. There are many possible mapping profiles[9] to transform relational data into RDF and OWL DL depending on specific requirements (Hert et al. 2011). In our work, minor enhancements to the default mapping profile proposed in the D2RQ (D2RQ 2012) works well with the target databases to carry all the manufacturing service information into OWL DL. Since the D2RQ's default mapping profile only uses RDF vocabulary and semantics, it is enhanced by specializing RDF vocabulary with OWL vocabulary (e.g., replace `rdf:Class` with `owl:Class`) and 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.

[9] 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.

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.

Figure 5 below shows an example of the transformation from a relational database table into 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.

Figure 5 should be inserted about here.

3.5 OPCs Identification

The purpose of the Ontology Pattern Correspondences (OPCs) Identification step is to select a unique ODP for a specific fragment of the source ontology. The OPCs identification process starts with establishing semantic links between entities and literals in the source ontology and the elements of the pattern library (by matching on their intended meaning). Semantic links can be *homogeneous* or *heterogeneous*. A homogeneous link is between the same type of entities and literals such as class-to-class, property-to-property, individual-to-individual, and literal-to-literal. A heterogeneous link is between different types of entities and literals such as class-to-individual, class-to-property and property-to-individual. Links maybe of any cardinality. For instance, a part size capability may be represented as a single concept with the literal value `"5 – 25 cm"` on the proprietary side, but it 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 (which relies on the notion of semantic similarity measure). A brief discussion of ontology matching is provided in the Related Works section. For the purpose of focusing on the canonicalization framework, let's assume that the user interactively establishes/disestablishes the semantic links with assistance from an ontology matcher while there is an underlying system utilizing the links information to suggest ODPs for a specific set of source ontology artifacts. It is at the user's discretion to correctly select an ODP for a particular set of ontology artifacts that results in an OPC. In the illustration below, we describe how this may occur.

The selected ODP within an OPC is called target ontology pattern. A given ODP can be the target for more than one set of source artifacts. For example, there can be multiple instance data that use the same design pattern. Therefore, the source ontology artifacts within the OPC are expected to be *representative ontology artifacts*. At the initiation of the OPC, some representative ontology artifacts are readily provided based on the semantic links. In some cases, however, the set of source ontology artifacts may be incomplete with respect to the requirements of the pattern. In the next stage – *OPCs identification completion* – all the necessary representative source ontology artifacts are identified by the user; and, structural differences between the target ontology pattern and the source ontology artifacts within the OPC are determined. If differences exist, subsequent canonicalization steps are necessary for that OPC. If there is no difference (i.e., they are structurally aligned), no further canonicalization step is necessary. It should be noted that the process of creating OPCs may not be linear and interaction with other ODPs can occur after a pattern transformation (i.e., certain entities maybe subjected to pattern transformation several times such as when one ODP suggests to turn an instance into a class and another ODP suggests such class to be a subclass of another class).

Figure 6 should be inserted about here.

OPCs Identification Illustration

Figure 6 shows an example source ontology. The semantic links, shown below, are established by the user between the source ontology and the pattern library. 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 shown in Figure 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 Figure 7). The canonicalization system can assist the user during this semantic relations inspection by identifying paths (starting from the shortest one) between the matching source ontology artifacts (in this case the `s:Supplier` and `s:ServiceCategory`). Once the user selects a suggested ODP, an ontology patterns correspondence is initiated to capture information related to the suggestion, such as `opc1` shown in Table 2. Notice that in the initial stage of the `OPC1`, parts of the representative artifacts are identified from the semantic links. They don't form a complete source ontology (graph) fragment (i.e., the s:Supplier and s:ServiceCategory are not connected).

Figure 7 should be inserted about here.

In the last stage – the OPC identification completion stage – the user identifies all the representative source ontology artifacts for opc1 and determines whether opc1 requires subsequent canonicalization steps. The resulting representative source ontology artifacts are shown in Table 3 that gives a complete

source ontology fragment. Notice that only `s:SupplierA` and `s:EDM` instances are identified as representative source ontology artifacts; other service categories and instances exist in the example source ontology, such as `s:SupplierB` and `s:Machining` (see Figure 6), that also match this pattern and need an application of this ODP. These other instances will be identified via the source ontology pattern created in the next canonicalization step. By analyzing the graph structures, user can determine that `OPC1` is an OPCs identification output, which 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:ServiceSubcategories`) 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`.

Table 2 should be inserted about here.

Table 3 should be inserted about here.

3.6 Source Ontology Patterns Generation

A source ontology pattern in an OPC is used to retrieve all pattern instances, each of which is a source ontology fragment containing a set of source ontology artifacts that will be transformed according to the respective target ontology pattern. In other words, a source ontology pattern is a set of parameterized representative source ontology artifacts within an OPC (this is analogous to a regular expression but it is an ontology graph). More specifically, a source ontology pattern consists of a signature and a set of binding expressions. The signature is a parameterized ontology structure. Binding expressions indicate the fixed part of the ontology structure by connecting parameters in the signature to the entities and literals in the source ontology artifacts. The parameters that are unbound are the variable part of the structure. Together the signature and binding expressions must be sufficient 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 from the source ontology. At the end of the source ontology patterns generation step, source ontology patterns are defined for each OPC output from the OPCs identification step.

Figure 8 should be inserted about here.

The source ontology patterns 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 expression can be generated, which the user can confirm. Specific binding expressions can be recommended based on the semantic links established in the OPCs identification step. That is, parameters in the source ontology signature that are linked to the fixed part of the target ontology pattern will also be suggested to be fixed (i.e., bounded by the binding expressions). The query

generation can be automated based on PATOMAT pattern-based ontology transformation described in the Related Works section (Svab-Zamazal et al 2009, Svab-Zamazal and Svatek 2011).

Figure 8 shows a graphical representation of an exemplary source ontology pattern, 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.

Figure 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.

Table 4 should be inserted about here.

Figure 9 should be inserted about here.

3.7 Transformation Rules Generation

In this process, a *transformation rule* (TR) is generated for each OPC. A TR 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 OPCs identification step, the 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, `TR1`, for `OPC1`. `OPC1` consists of source and target ontology pattern shown in Figure 8 and Figure 4. It requires 10 PTRs.

Table 5 should be inserted about here.

3.8 Pattern Transformation

Pattern transformation executes 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 canonicalized proprietary MSC model is expected to be structurally aligned 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, pattern instances detection and transformation rule application. The whole process can be automated based on the aforementioned PATOMAT work.

The pattern instances 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 Figure 9 will retrieve all pattern instances for `OPC1`. Two pattern instances should be returned for the source ontology in Figure 6.

The transformation rule application process applies the transformation rule 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 transformation rules on the source ontology, the canonicalized proprietary MSC model is obtained as the final output. Next, canonicalization of an exemplary proprietary MSC data model is provided to validate the applicability of this framework.

4 Canonicalization Example

In this section, the canonicalization framework is demonstrated on a realistic example. We first describe inputs to the canonicalization process including a pattern library and a proprietary MSC data model which 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 mapping in section 5.

4.1 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`. Figure 10 illustrates the definitions of these ODPs.

Figure 10 should be inserted about here.

4.2 Proprietary MSC Data Model

Figure 11 below shows a set of relational tables in a proprietary MSC data model. It represents how supplier, service, service category, and part length capability are represented and related in a proprietary way.

Figure 11 should be inserted about here.

4.3 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. Figure 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.

Figure 12 should be inserted about here.

Transformation of a table with foreign keys involves creating a class, named individuals, object properties, data properties, and assertions. Figure 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 records in the `Supplier` and `Service` tables as shown in the figure.

Figure 13 should be inserted about here.

Figure 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 Figure 11. The rest of the canonicalization illustration will be based on this data.

Figure 14 should be inserted about here.

4.4 OPCs Identification

At OPCs identification time, first, the semantic links are established between the source ontology and the pattern library as shown below. For example, the user has linked `s:EDM` and `s:Moldmaking` with the `p:ServiceCategory` as they semantically mean service category in the proprietary manufacturing service capability model. With these semantic links and the ODPs' binding expressions, the ODPs that are related to these terms are retrieved and the OPCs are initialized as shown in Table 6. Note that the `OPC3` is initialized not just by using the semantic links but also using logical inference, 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 Figure 10, the `Service-Categorization` ODP illustrates that an instance of `p:Service` is an instance of `p:ServiceCategory` as well. Thus, `s:EDM` is indirectly linked 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 that the logical structure of each source ontology artifact differs from its corresponding ODP. Thus, `OPC1, OPC2, OPC3, OPC4 and OPC5` are the OPCs output from the OPCs identification on which subsequent canonicalization steps will be performed.

Table 6 should be inserted about here.

Table 7 should be inserted about here.

4.5 Source Ontology Patterns Generation

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

Figure 15 should be inserted about here.

4.6 Transformation Rules Generation

With the source and target ontology patterns captured in the OPCs, pattern transformation rules (PTRs) can be created for each OPC. Figure 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. `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.

Figure 16 should be inserted about here.

Table 8 should be inserted about here.

Transformation rules for OPC2, OPC3 and OPC4 are denoted by TR2, TR3, and TR4, respectively. Their PTRs only need to deal with structural pattern detections and entity transformations similar to those of OPC1 and consequently can be created in the same way. PTRs for OPC5, however, need to additionally deal with literal value pattern detections and transformations. This is because the LengthCapability ODP has two data properties including p:hasMin and p:hasMax. However, the SP5 source ontology pattern has one data property that represents the part length capability min and max values with a single literal value such like "6cm - 48cm". Figure 17 illustrates the situation.

Figure 17 should be inserted about here.

To deal with this situation, a literal value pattern is defined with the following regular expression.

- ([0-9]+)cm - ([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 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 value and is assigned to the variable s:G2. This literal value pattern detection is used for PTR5.3 and PTR5.4 as indicated by their usages of s:G1 and s:G2 in Table 10.

Table 9 should be inserted about here.

4.7 Pattern Transformation

The pattern transformation process is a final step of the canonicalization and it executes transformation rules on the source ontology for each OPC. Table 11 below summarizes the current state of the OPCs that provide sufficient information to execute the pattern transformation.

Table 10 should be inserted about here.

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.

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 Figure 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 Figure 14.

Figure 18 should be inserted about here.

Table 11 should be inserted about here.

The pattern transformation rules application sub-process applies PTRs on the retrieved entities and literals. Table 12 shows the result of applying PTRs in `TR1` (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). Figure 19 shows the final output of the canonicalization process from executing pattern transformation on `OPC1` to `OPC5`.

Table 12 should be inserted about here.

Figure 19 should be inserted about here.

5 Analysis

This section provides qualitative and quantitative analyses to demonstrate that canonicalization enables simplified OWL mapping axioms, shorter reasoning time, and better semantic mediation results. Specifically, the analyses show the benefits canonicalization can provide to the last step, *OWL Mapping & Inference*, in the OWL-based semantic mediation process shown in Figure 1. It should be noted, however, that the analyses are not intended to show, nor do they prove, that the whole semantic mediation process is shortened with canonicalization.

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 0, 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 the transformations of the non-canonicalized schema and the corresponding data that are described in Section 4. The next two subsections provide a detailed discussion of these analyses.

5.1 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. Since the queries use only terminologies and structures from the reference MSC ontology to retrieve information encoded in the proprietary terminologies and structures, they verify what semantic mediation has occurred.

MSC information related to `Supplier_3` in the proprietary MSC data model shown in Figure 11 is used for this illustration. The MSC information has two features including the service category and part length capability. The queries `Q1` and `Q2` in Figure 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`.

Figure 20 should be inserted about here.

Figure 21 should be inserted about here.

First, we analyze Case 1: OWL mapping between a canonicalized proprietary MSC model and the reference MSC ontology. On the left side of Figure 21 is the canonicalized proprietary MSC model of `Supplier_3` information 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 successfully return the desired results. Notice that both the subjects and objects of the mapping axioms are simply a single class or property name without the 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 non-canonicalized proprietary MSC model (i.e., source ontology) and the reference MSC ontology. Figure 22 illustrates this case. In this analysis, it is assumed that the non-canonicalized proprietary MSC model is syntactically and automatically transformed from its RDB representation. This is shown on the left side of the figure with `Supplier_3` information; and on the right side is the reference MSC ontology. Unlike the previous case, simple class-to-class and property-to-property mapping axioms are insufficient to enable `Q1` and `Q2` in this case. Specifically, there are two issues and they are highlighted in the shaded areas connected by the double arrow lines in the figure. First there is a need to map between differing OWL ontology artifact types, specifically from a literal `"6cm to 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, this is not mappable within OWL DL.

The other issue is that there is a need to bridge the views between two different structures representing the concept of supplier having an EDM manufacturing service. (Also, note that the non-canonicalized model has a rather more complex structure than that in the reference MSC ontology.) Such bridging is necessary to enable `Q2`. This is achieved by the mapping class technique (Kulvatunyou et al. 2013). The required mapping class `EDMSupplier` is shown in Table 13. 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, mapping class axioms are the `A1` and `A2` axioms in Table 13. Although these mapping class axioms enable `Q2` to return the desired results, the query behavior associated with mapping class approach is somewhat restricted. The reason is that the bridge between the views is limited to the view (i.e., class expressions) provided in the axioms. For example, the `EDMSupplier` mapping class in `A1` bridges view 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, as can be seen that `A1` and `A2` only address one service category.

All-in-all, this analysis shows that canonicalization before OWL mapping can circumvent OWL DL limitations, can simplify the mappings by avoiding the need for OWL class expression in the mapping class, and can simplify the mapping maintenance by reducing the number and complexity of mapping axioms. The quantitative analysis in the next section will show another benefit of canonicalization by demonstrating that the growing number of mapping classes due to the growing number of service categories has a negative impact on the OWL inference time.

Figure 22 should be inserted about here.

Table 13 should be inserted about here.

5.2 Quantitative Analysis

The quantitative analysis compares the OWL inference time as influenced by the two types of mapping to reference ontology: using canonicalized vs. non-canonicalized proprietary MSC models. Specifically, we study the impact of the number of mapping classes on the amount of time required to complete the OWL DL reasoning process required by the OWL-based semantic mediation. The reasoning process includes four tasks including loading, consistency checking, classification, and realization as described in (MINDSWAP 2012). Brief descriptions of these tasks are provided below. All the reasoning are performed with Pellet version 2.2.2 (Clark and Parsia 2012) on a Dell Precision WorkStation T5500 with Intel Xeon E5504 2.00GHz (8 CPUs) and 2GB of main memory.

Reasoning task:

1. **Loading** loads an RDF model to the reasoner.
2. **Consistency checking** ensures that an ontology does not contain any contradictory facts. OWL Abstract Syntax & Semantics document provides a formal definition of ontology consistency that the reasoner uses. In DL terminology, this is the operation to check the consistency of an ABox with respect to a TBox.
3. **Classification** computes the subclass relation between each named class to create the class hierarchy.
4. **Realization** finds the most specific classes that an individual belongs to; in other words, it computes the direct types for each of the individuals.

In this analysis, only part of the MSC information from the previous section that is related to the supplier's service declaration is used. We also show, in this case, that even if a human applies additional rule-based transformation to the syntactically transformed MSC information, it may still not be structurally aligned with the reference ontology and cause a reasoning inefficiency. Figure 23 shows this additional lift-up based on the proprietary view. It shows that 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. This modeling pattern, which represents service category as an instance of the service class, is abstracted out with polymorphic names for several declarations of service capabilities in the left side of Figure 25. 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 Figure 24 and Figure 25, respectively.

Figure 23 should be inserted about here.

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`. Generally, 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`).

Figure 24 should be inserted about here.

Figure 25 should be inserted about here.

Table 14 shows the reasoning times in the canonicalization case. It can be seen that the increase in the number of service categories has little impact on the consistency checking and classification times, while minor increases can be observed 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 Figure 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 significant role in a practical deployment of OWL DL-based semantic mediation when there are structural conflicts.

Table 14 should be inserted about here.

Table 15 should be inserted about here.

Figure 26 should be inserted about here.

6 Related Works

There are five steps in the framework for canonicalization shown in Figure 2 including the transformation, OPCs identification, source ontology patterns generation, transformation rules generation and pattern transformation. In this section, we discuss the works that are relevant to enabling these steps.

The transformation step transforms heterogeneous syntaxes of data into a common RDF syntax using OWL DL vocabulary. Currently, there are abundantly many tools to support RDB-to-RDF transformations. The W3C RDB2RDF Incubator group has presented a survey on these tools such as D2RQ, 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, and D2RQ 2012). All-in-all, we have found that the D2RQ is capable of supporting the automatic transformation via the mapping profile outlined in section 3.1.

The OPCs identification process starts with establishing semantic links between entities and literals in the source ontology and the archetypical entities and literals in the pattern library. This stage requires ODPs that have been developed beforehand. With regard to the ODPs development, there have been recent research works that were supported by the NeOn project[10]. In particular, the deliverable D2.5.1 (Presutti et al. 2008) provides a methodology to create ODPs. In addition, works related to pattern library includes Suárez-Figueroa et al. (2007) and Presutti et al. (2008). Suárez-Figueroa et al. (2007) have presented a general template for describing ODPs and an initial repository of OWL-based ODPs. Presutti et al. (2008) have classified ODPs into structural, correspondence, reasoning, presentation, and lexico-syntactic and content and have also specified functional requirements of a repository for ODPs. A repository following such specification has been implemented on-line at the Ontology Design Patterns wiki page[11].

Manually establishing the terminological links in the OPC identification can become cumbersome when the information sources in the source ontology and pattern library are large. Hence, ontology matching algorithms which suggest candidates for terminological links can be useful. Shvaiko and Euzenat (2011) have summarized and analyzed works in ontology matching in the past decade. Applications of ontology matching algorithms have been largely focused on achieving full ontology mapping or alignment, which the authors have indicated that there are still open issues, particularly the ability to match across entity types (i.e., to match across structural conflicts). However, matching task such like in the OPCs identification outlined in our canonicalization framework calls for terminology matches without an indication of structural relationship. Therefore, we hypothesize that ontology matching algorithms would be suitable to the OPCs identification task. This will be a topic of future work.

These last three canonicalization steps can be collectively supported by a *pattern-based ontology transformation* solution. Key enablers for the pattern-based ontology transformations are pattern transformation definition, pattern instances detection engine, and pattern transformation engine. PATOMAT project (Svab-Zamazal et al 2009, Svab-Zamazal and Svatek 2011) provides workable methods and tools that include these key enablers. It provides a well-defined XML schema for the pattern transformation definition (including pattern definitions and transformation rules). For pattern instances detection engine, PATOMAT provides the functionality to generate SPARQL query from the pattern transformation definition. Its pattern transformation engine uses OPPL application interface (OPPL 2012) for pattern transformation. PATOMAT also has the TPEditor component which is a graphical user

[10] http://www.neon-project.org/
[11] http://www.ontologydesignpatterns.org

interface-based editor of source and target ontology patterns and associated transformation rules. The output of the TPEditor is an XML instance file conformed to the PATOMAT's XML schema for pattern transformation definition. Our analysis has indicated that some enhancements to PATOMAT functions are necessary to fully support the proposed canonicalization framework. First, PATOMAT does not deal with the representative artifacts which represent the varying parts of the source ontology pattern. Thus, even in the case that there are multiple pattern instances that use the same source ontology pattern, recursive transformation rule applications on those pattern instances do not occur. Second, PATOMAT does not provide any computer-assisted method to generate source ontology patterns and to retrieve reusable target ontology pattern. All the patterns need to be defined manually at present. Lastly, the literal values may contain important concepts within the resulting RDB-to-OWL transformation. However, there is no facility to deal with literal value pattern detections and transformations at present.

7 Conclusion and Future Works

In acquiring manufacturing services, the customer is confronted with a number of diverse supplier databases with different structures, terminologies, and query languages. Using a semantic mediation approach – providing a common reference ontology for accurately mapping source information into that and other forms – can simplify the customer's problem. This in-turn enables easier reconfiguration of a supply chain, leading to a better supply chain performance.

We have shown in earlier work that the OWL language is suitable for capturing both the reference ontology and the underlying ontologies for the supplier data schemas. It has also shown that an OWL reasoner can be used to perform the semantic information transformations involved in answering the customer queries.

OWL-based semantic mediation that seeks to address all kinds of semantic conflicts in a single transformation step requires mapping statements that use complex expressions. This makes the use of the DL reasoner for the transformations complex and computationally expensive, and it does not always completely render the source information.

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; and the second step addresses other conflicts via OWL DL mapping axioms. We presented a detailed framework to achieve the canonicalization, and we demonstrated its applicability using a realistic example of a proprietary manufacturing service capability database.

We presented qualitative and quantitative analyses of the impact of canonicalization on the mapping task. The qualitative analysis shows that canonicalization can circumvent OWL DL limitations, simplify the OWL mapping axiom by avoiding the need for complex OWL class expressions in the mapping axioms, and simplify the mapping maintenance by reducing the number and complexity of mapping axioms. The quantitative analysis shows that OWL reasoning time grows cubically when OWL DL axioms

are used to resolve a common type of structural conflict, while the reasoning time for the canonicalization approach grows linearly.

The primary contribution of this research is the formalization of the canonicalization process in a framework that encompasses complementary work in syntactical data transformation, ontology design patterns, ontology matching, and pattern-based ontology transformation. In addition, the paper outlines a novel approach to representing reusable conceptual ontology design patterns and capturing the ontology pattern correspondences (OPCs) in the source ontology.

In terms of future work, we are developing an implementation to support the proposed framework. This includes: the pattern library storage which supports the proposed conceptual ontology design pattern representation; a computer-assisted OPC identification environment that employs ontology-matching algorithms; and a software component that help the user manage the ontology pattern correspondence throughout the canonicalization process. Enhancements to the existing pattern transformation engine are also needed as outlined in the related work. Our further work also includes developing 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.

References

Ameri F. and Dutta, D. (2006), An Upper Ontology for Manufacturing Service Description, *ASME International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*, Philadelphia, 10 – 13.

Bizer, C. (2003), D2R MAP : A Database to RDF Mapping Language, *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, *Proceedings of 3rd International Semantic Web Conference* (Hiroshima, Japan, 2004), S.A. McIlraith, D. Plexousakis, F. van Harmelen, Eds., Springer, Lecture Note in Computer Sciences 3298.

Clark and Parsia LLC (2012), Pellet: OWL 2 Reasoner for Java version 2.2.2, available online at http://clarkparsia.com/pellet/, accessed September 2012.

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

D2RQ framework version 0.8.1, Accessing Relational Databases as Virtual RDF Graphs, available online at http://d2rq.org/, accessed September 2012.

Gangemi, A. (2005), Ontology Design Patterns for Semantic Web Content, International Semantic Web Conference, Springer-Verlag Lecture Note in Computer Sciences 3729, pp. 262–276.

Hert, M., Reif, G., Gall, H. C. (2011), A Comparison of RDB-to-RDF Mapping Languages, *Proceedings of the International Conference on Semantic Systems*, Graz, Austria, 2011.

Kulvatunyou, B.S., Ivezic, N., Lee, Y., Shin, J. (2013), An Analysis of OWL-based Semantic Mediation Approaches to Enhance Manufacturing Service Capability Models, to appear in *International Journal of Computer Integrated Manufacturing*, available online at http://www.nist.gov/publication-portal.cfm#.

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

MINDSWAP - Maryland Information and Network Dynamics Lab Semantic Web Agents Project (2012), *Pellet Performance Report*, available online at http://www.mindswap.org/2003/pellet/performance.shtml, accessed September 2012.

OPPL - Ontology Pre-Processor Language version 2 (2012), available at http://oppl2.sourceforge.net/, accessed September 2012.

Park, J., and Ram, S. (2004), Information Systems Interoperability: What Lies Beneath?, *ACM Transactions on Information Systems*, Vol. 22(4), pp. 595-632.

Presutti, V., Gangemi, A., David, S., Aguado de Cea, G., Suárez-Figueroa, M.C., Montiel-Ponsoda, E., Poveda, M. (2008), *NeOn Project Delivery - D2.5.1. A Library of Ontology Design Patterns: reusable solutions for collaborative design of networked ontologies*.

RFC3987 - Request for Comments: 3987 (January 2005), *Internationalized Resource Identifiers (IRIs)*, available online at http://www.ietf.org/rfc/rfc3987.txt.

Satya, S.S., Halb, W., Hellmann, S., Idehen, K., Thibodeau, T., Auer, S., Sequeda, J., and Ezzat, A. (2009), *A Survey of Current Approaches for Mapping of Relational Databases to RDF, W3C RDB2RDF Incubator Group*, available online at http://www.w3.org/2005/Incubator/rdb2rdf/RDB2RDF_SurveyReport.pdf

Sheth, A. P., and Kashyap, V. (1992), So far (schematically), yet so near (semantically), *Proceedings of the IFIP WG2.6 Database Semantics Conference on Interoperable Database Systems* (DS-5, Lorne, Victoria, Australia, Nov. 16–20), Hsiao, D. K., Neuhold, E. J., and Sacks-Davis, R., Eds., 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, available online at https://smart-process-manufacturing.ucla.edu/about/news/Smart%20Manufacturing%206_24_11.pdf.

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

Svab-Zamazal O. and Svatek V. (2011), OWL Matching Patterns Backed by Naming and Ontology Patterns, *Znalosti, 10th Czecho-Slovak Knowledge Technology Conference*, Stara Lesna, Slovakia.

Tsinaraki C., Polydoros, P., and Christodoulakis S. (2004), Interoperability support for Ontology-based Video Retrieval Applications, *Image and Video Retrieval*. Springer LNCS 3115, pp. 582-591.

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

W3C - World Wide Web Consortium (2004a), SWRL: A Semantic Web Rule Language, May 21, 2004, available online at 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, available online at http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

W3C - World Wide Web Consortium (2004c), XML Schema, Parts 0, 1, and 2 (Second Edition), October 28, 2004, available online at 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, available online at 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, available online at http://www.w3.org/TR/swbp-specified-values/

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

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

W3C - World Wide Web Consortium (2009a), OWL 2 Web Ontology Language, October 27, 2009, available online at 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, available online at 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, available online at http://www.w3.org/TR/owl2-manchester-syntax/.

Ye, Y., Yang, D., Jiang, Z., and Tong, L. (2007), An Ontology-Based Architecture for Implementing Semantic Integration of Supply-Chain Management, *International Journal of Computer Integrated Manufacturing*, Vol. 21(1), pp. 1–18.

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

A Framework to Canonicalize Manufacturing Service Capability Models

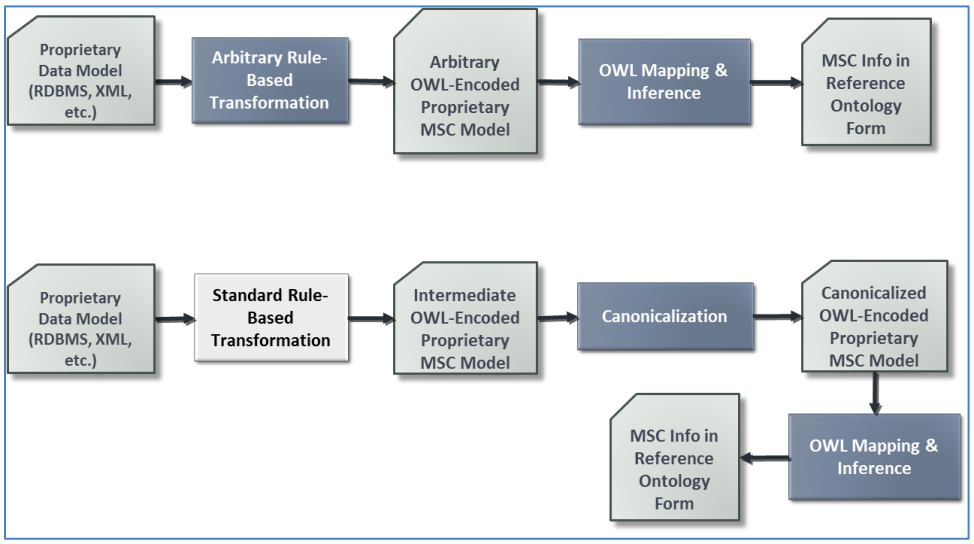


Figure 1: Typical (top) and proposed (bottom) process to OWL-based semantic mediation between proprietary MSC data models and Reference ontology (dark grey boxes typically require human involvement and are not fully automated)

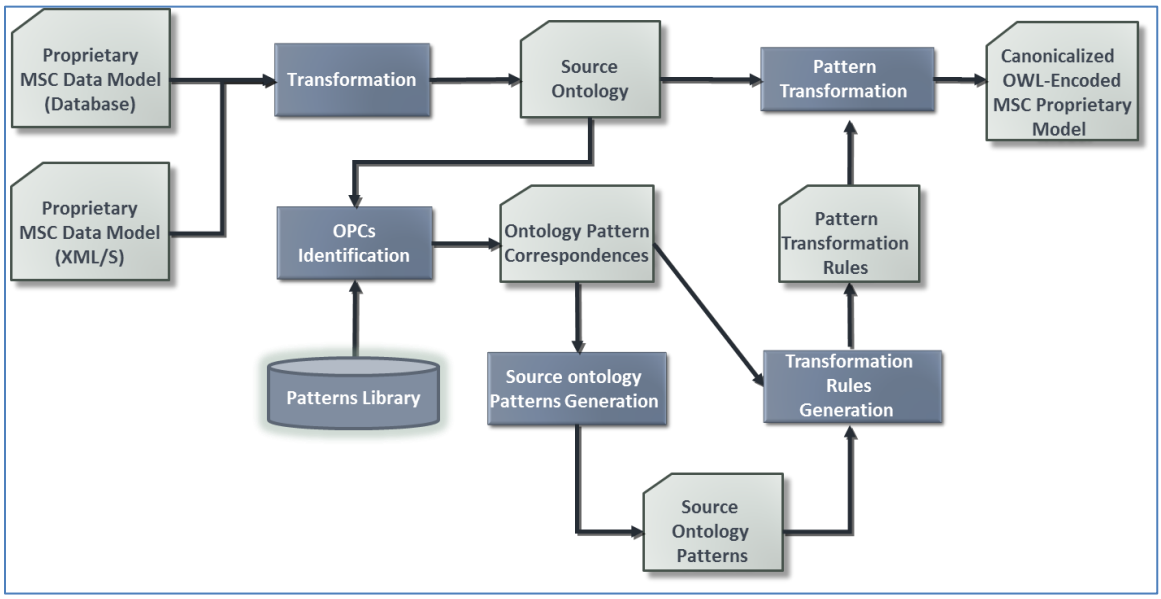


Figure 2: Canonicalization framework

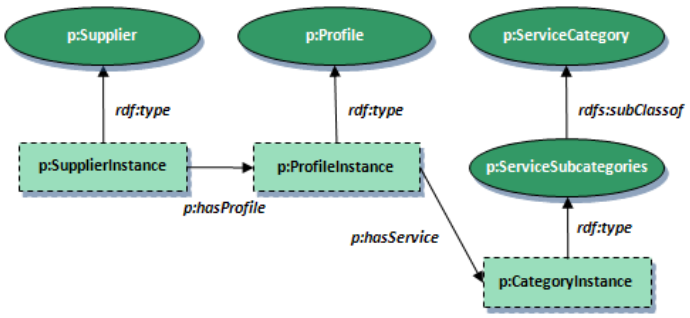


Figure 3: Archetypical ontology of the Supplier-Service ODP

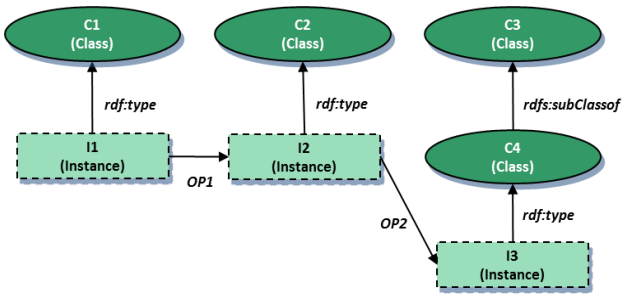
ODP Name/Description	ODP Signature	Binding Expression
Supplier-service: Supplier declares a service it provides and models a semantic service category as a (descendant) subclass of the abstract ServiceCategory class		C1 = p:Supplier C2 = p:Profile C3 = p:ServiceCategory OP1 = p:hasProfile OP2 = p:hasService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

Figure 4: Supplier-Service ODP example

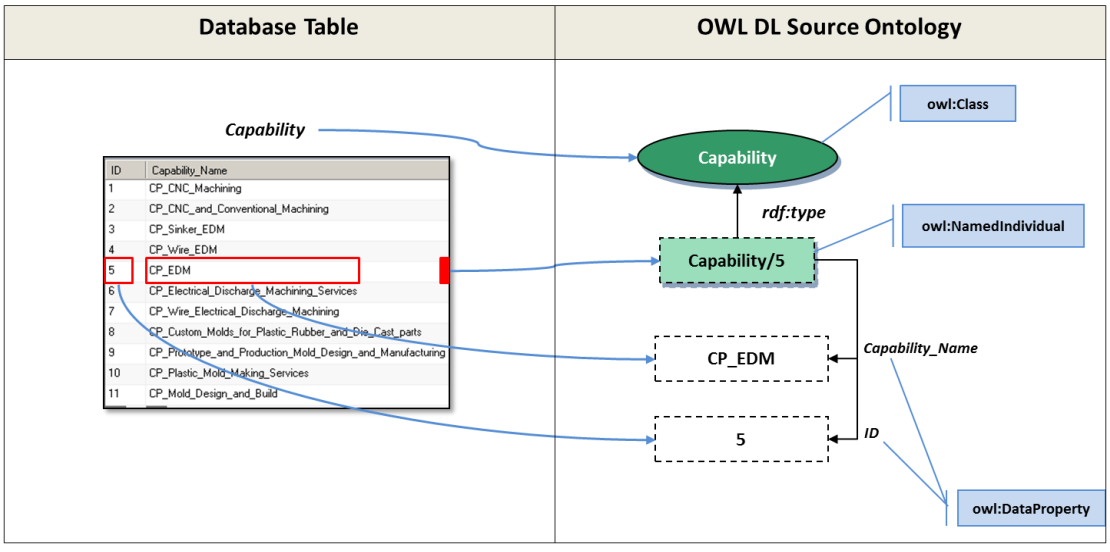


Figure 5: Transformation example

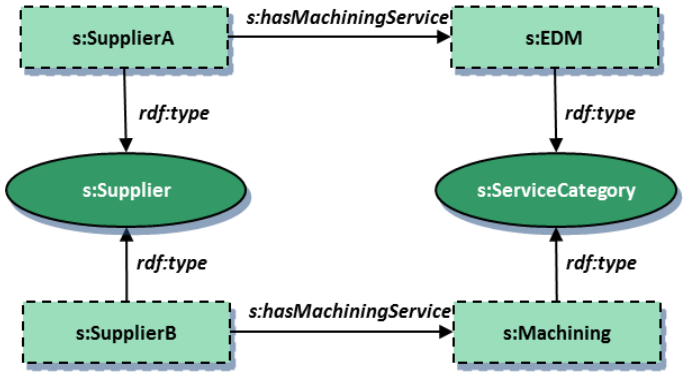


Figure 6: An example source ontology

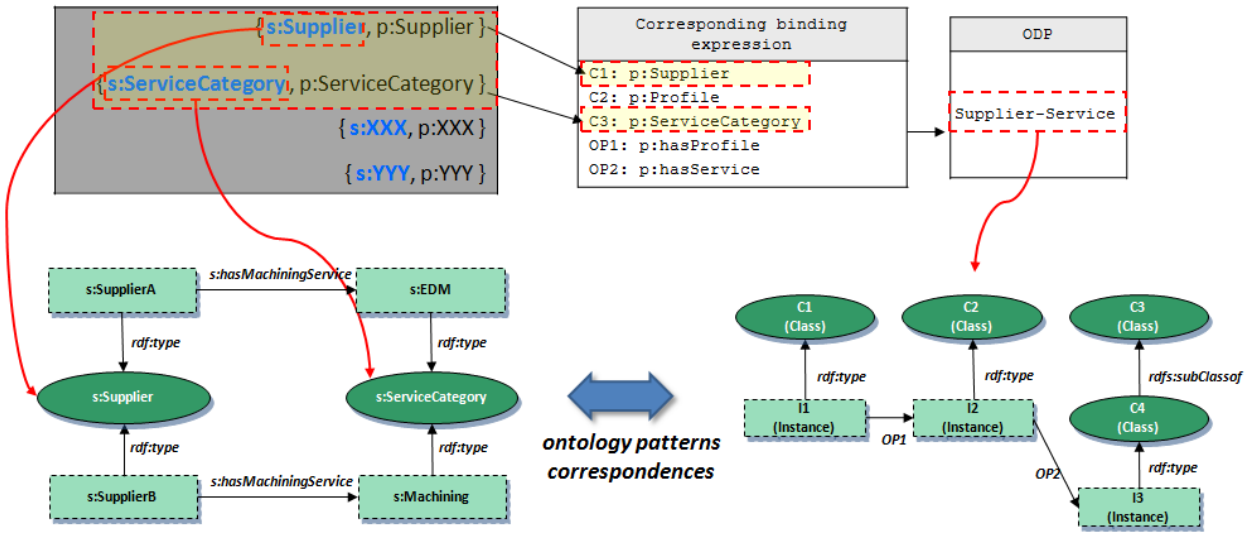


Figure 7: OPC identification process

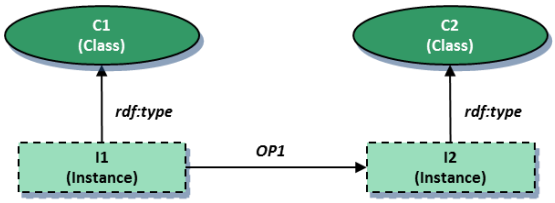
ID	Source Ontology Signature	Binding Expression
SP1		C1 = s:Supplier C2 = s:ServiceCategory OP1 = s:hasMachiningService **Unbounded Entities:** I1, I2

Figure 8: Graphical representation of the source ontology pattern for OPC1 (Table 3Error! Reference source not found.)

```
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 .
}
```

Figure 9: Source ontology pattern SPARQL query

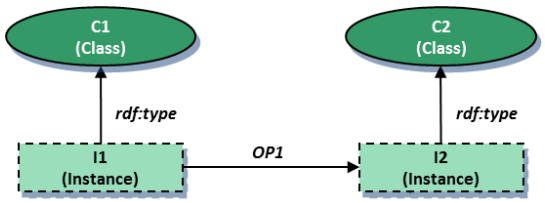
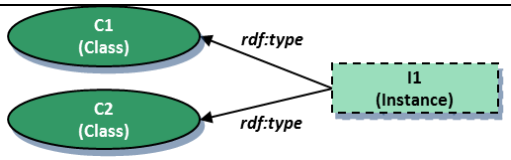
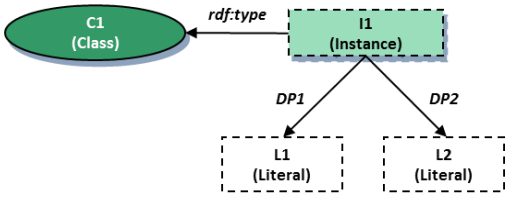
ODP Name/Description	Signature	Binding Expression
Supplier-Service: Supplier declares a service it provides		C1 = p:Supplier C2 = p:Service OP1 = p:hasService **Unbounded Entities**: I1, I2
Service-LengthCapability: A service declares a length capability value range, e.g, part envelop size, min/max acceptable diameter		C1 = p:Service C2 = p:LengthCapability OP1 = p:hasLengthCapability **Unbounded Entities**: I1, I2
Service-Categorization: Classify a service into a service category		C1 = p:Service C2 = p:ServiceCategory **Unbounded Entities**: I1
LengthCapability: A length capability representation, e.g., part envelop size, min/max acceptable diameter		C1 = p:LengthCapability DP1 = p:hasMax DP2 = p:hasMin **Unbounded Entities**: I1, L1, L2

Figure 10: Pattern library

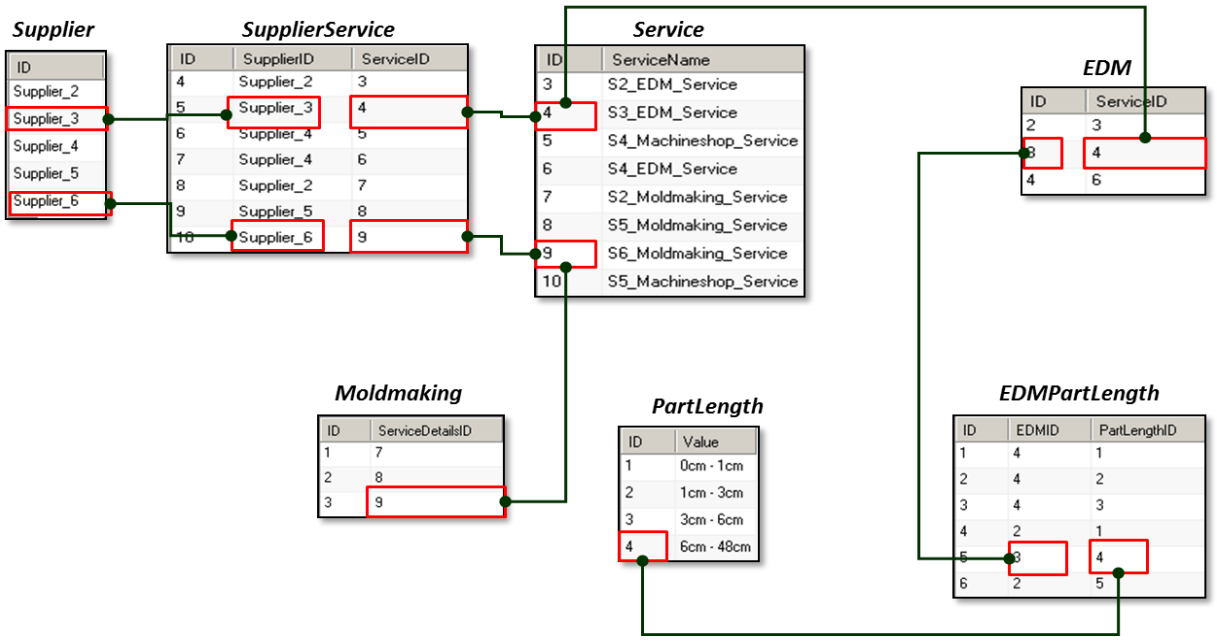


Figure 11: Proprietary manufacturing service capability data model

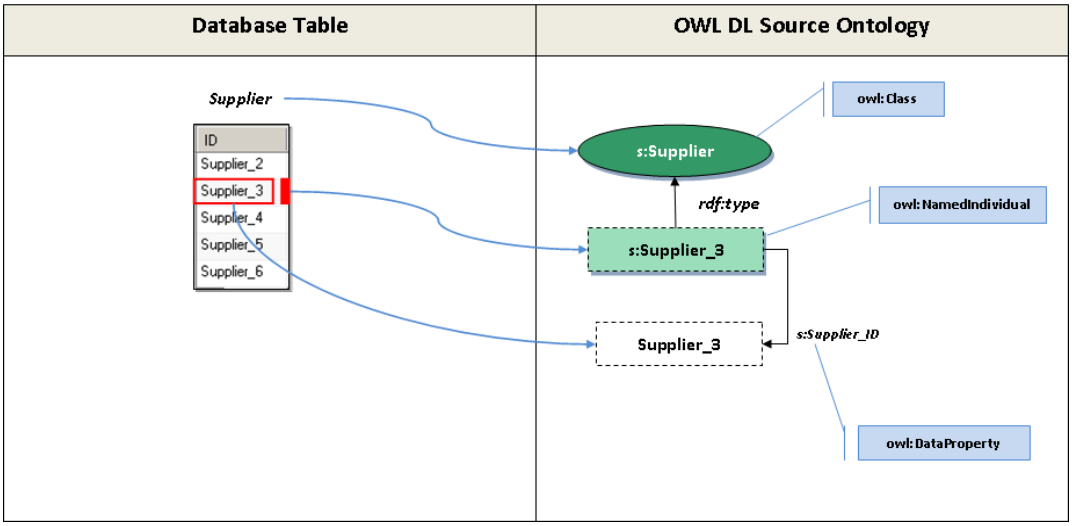


Figure 12: Standard rule-based OWL DL encoding of the `Supplier` table

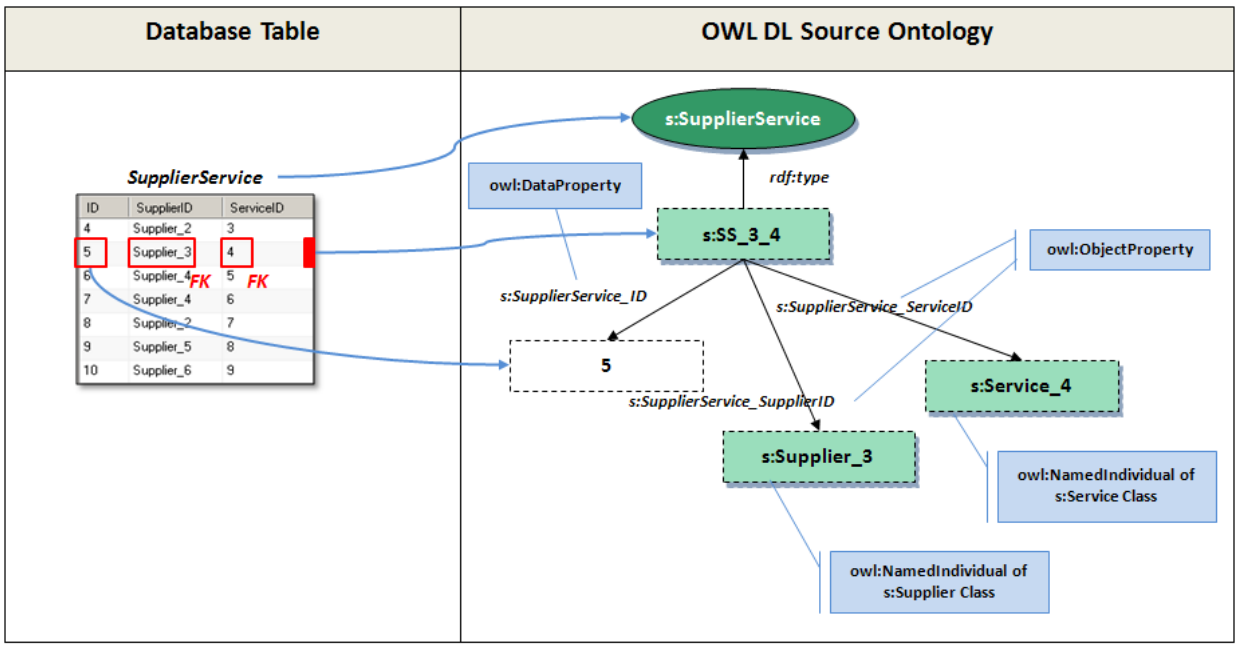


Figure 13: Standard rule-based OWL DL encoding of the `SupplierService` table

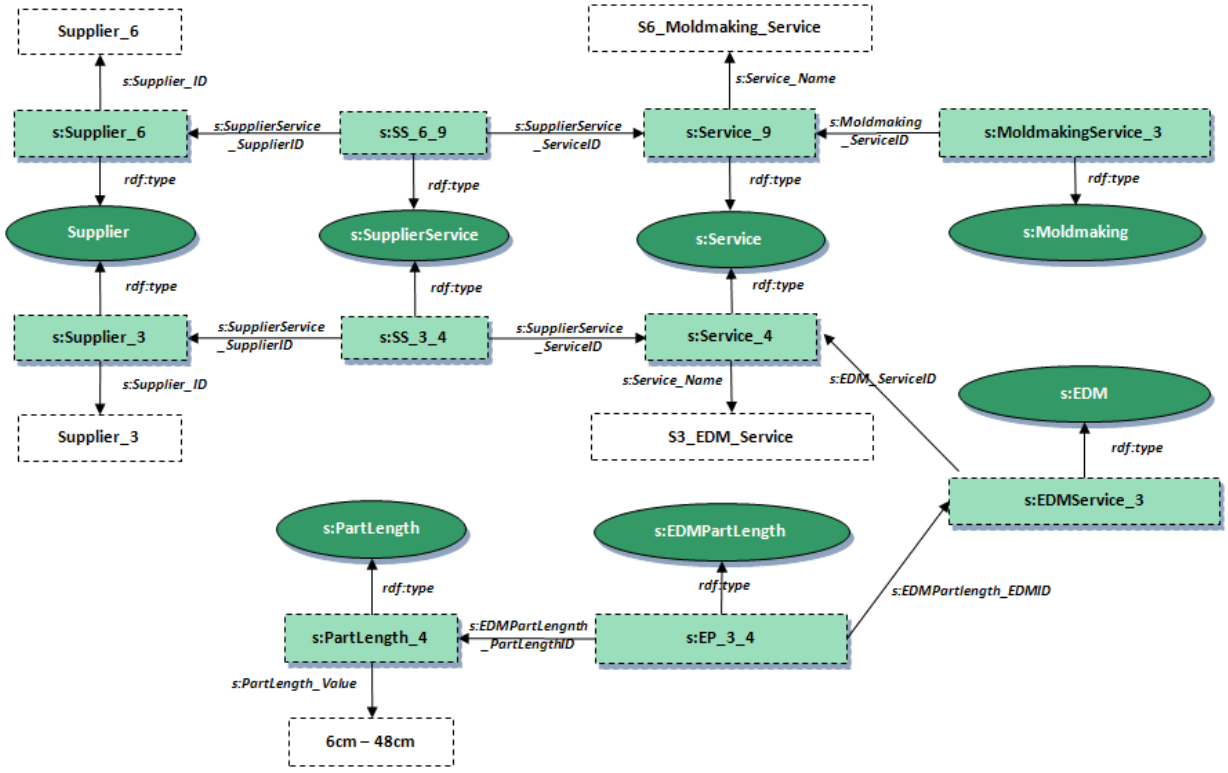


Figure 14: Source ontology

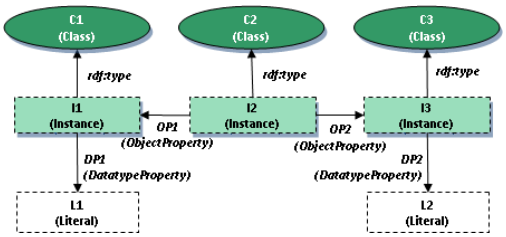
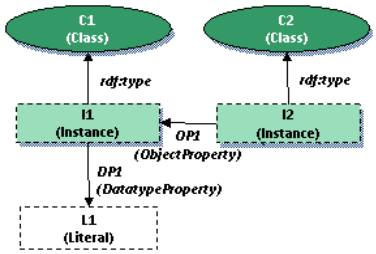
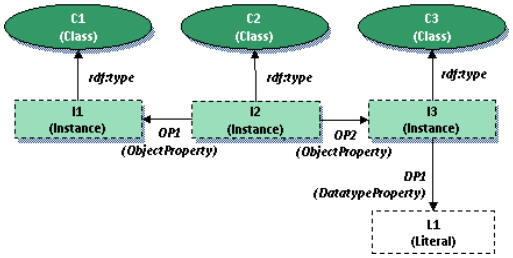
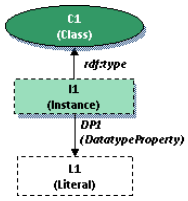
ID	Source Ontology Signature	Binding Expression
SP1		C1 = s:Supplier C2 = s:SuplierService C3 = s:Service OP1 = s:SupplierService_SupplierID OP2 = s:SupplierService_ServiceID DP1 = s:Supplier_ID DP2 = s:Service_Name **Unbounded Entities**: I1, I2, I3, L1, L2
SP2		C1 = s:Service C2 = s:EDM OP1 = s:EDM_ServiceID DP1 = s:Service_Name **Unbounded Entities**: I1, I2, L1
SP3		C1 = s:Service C2 = s:Moldmaking OP1 = s:Moldmaking_ServiceID DP1 = s:Service_Name **Unbounded Entities**: I1, I2, L1
SP4		C1 = s:EDM C2 = s:EDMPartLength C3 = s:PartLength OP1 = s:EDMPartlength_EDMID OP2 = s:EDMPartLengnth_PartLengthID DP1 = s:PartLength_Value **Unbounded Entities**: I1, I2, L1
SP5		C1 = s:PartLength DP1 = s:PartLength_Value **Unbounded Entities**: L1

Figure 15: Graphical representations of the source ontology patterns

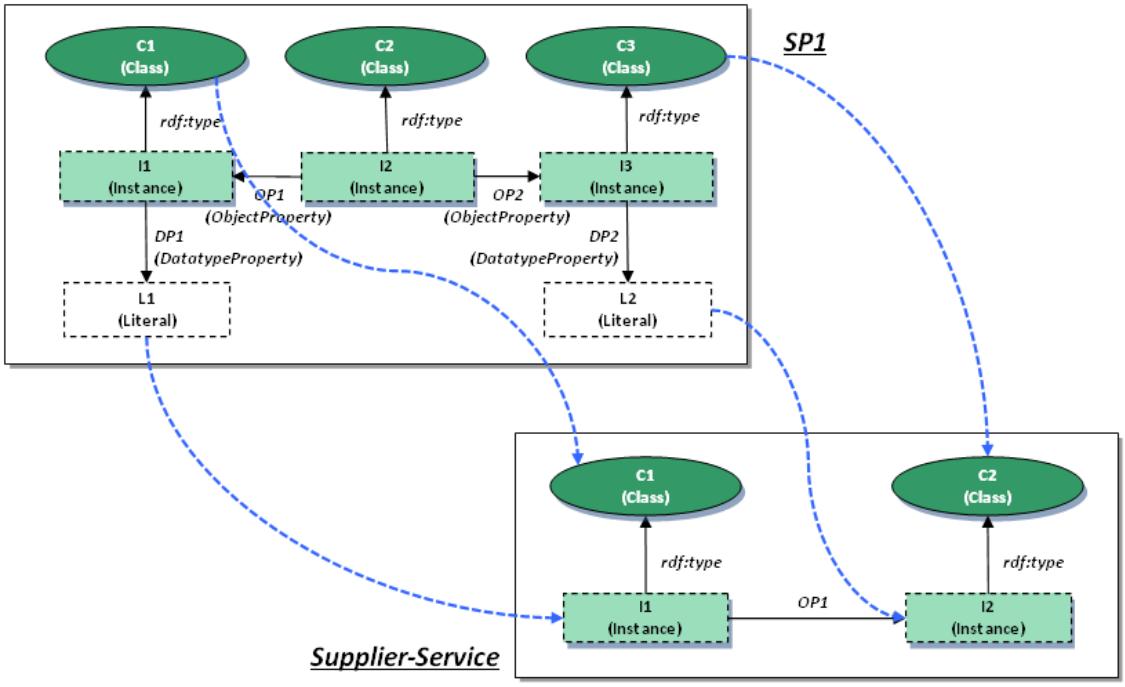


Figure 16: PTRs between the source and target ontology patterns in OPC1

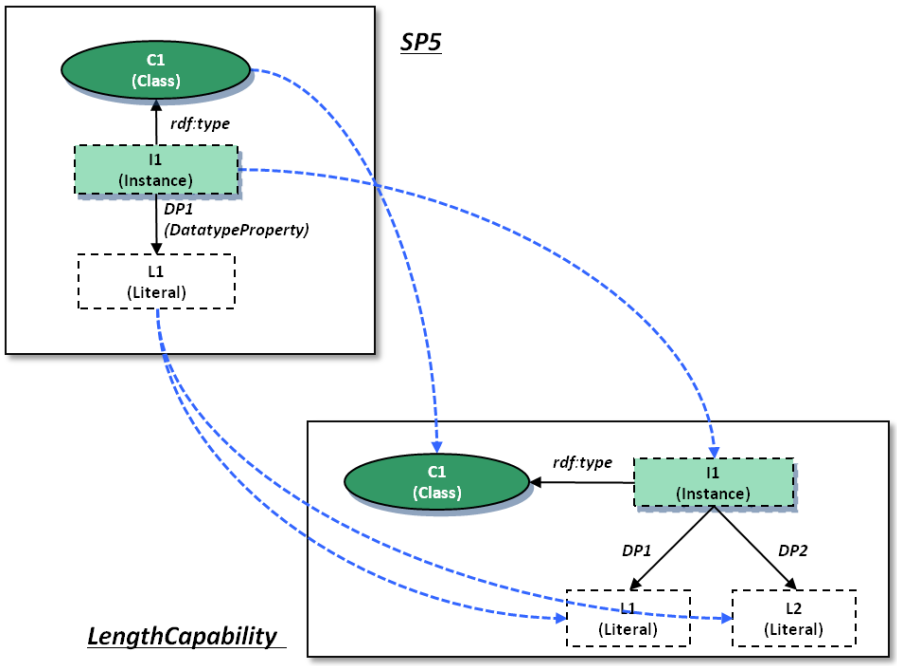


Figure 17: Literal value pattern detection

```
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 .
    }
```

Figure 18: SPARQL query generated from SP1

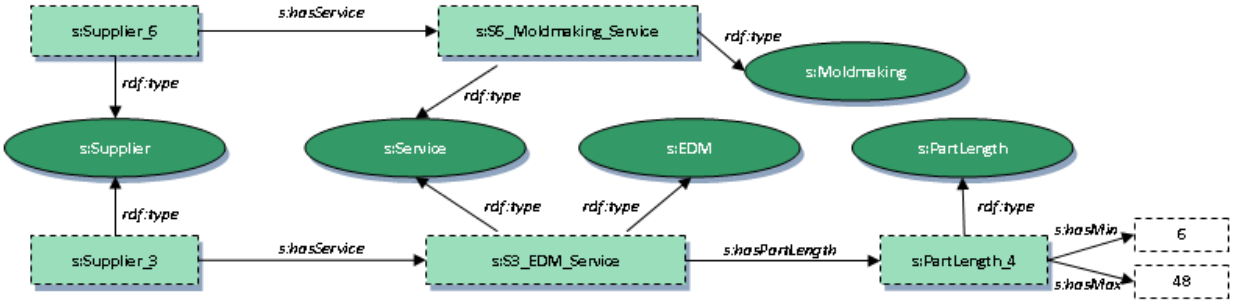


Figure 19: Canonicalized proprietary MSC model

```
Note: Terms in the reference MSC ontology are denoted by the prefix, 'mo'.

Q1: Identify suppliers having the ElectroDischargeMachiningService which has minimum 6cm
and maximum 48cm as part length capability value (using only terms from the reference MSC
ontology.)


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
reference MSC ontology.)

OWL DL Query => mo:Supplier and mo:hasService some mo:ElectroDischargeMachiningService
```

Figure 20: Desired query behavior to demonstrate the OWL-based semantic mediation

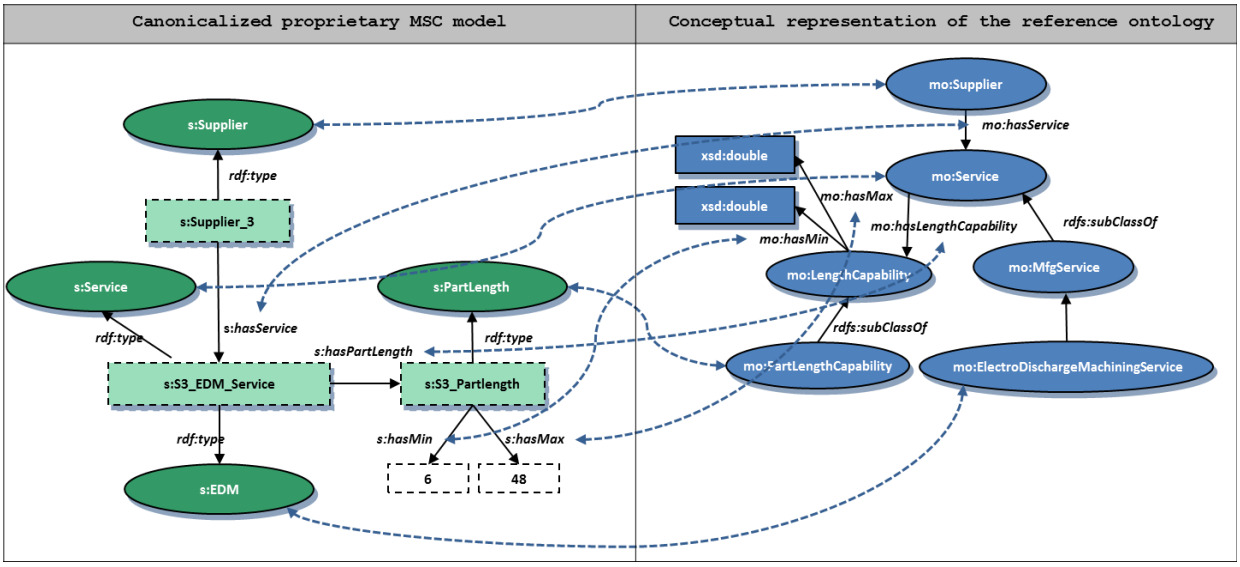


Figure 21: Qualitative analysis of OWL mapping after canonicalization

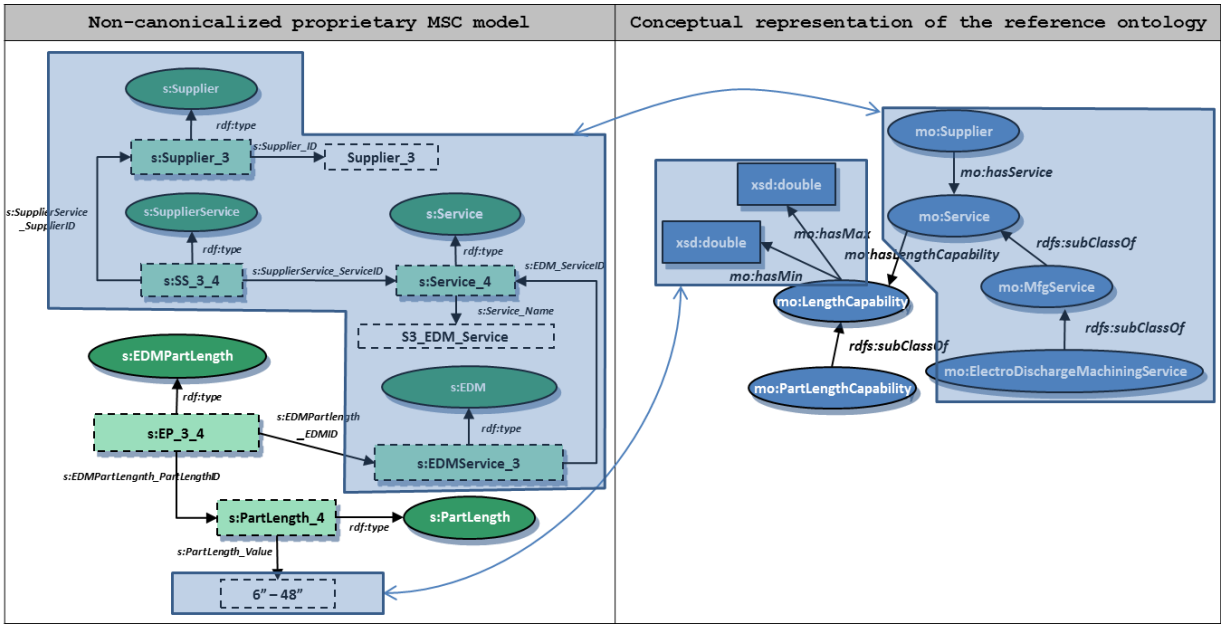


Figure 22: Qualitative analysis of OWL mapping without canonicalization

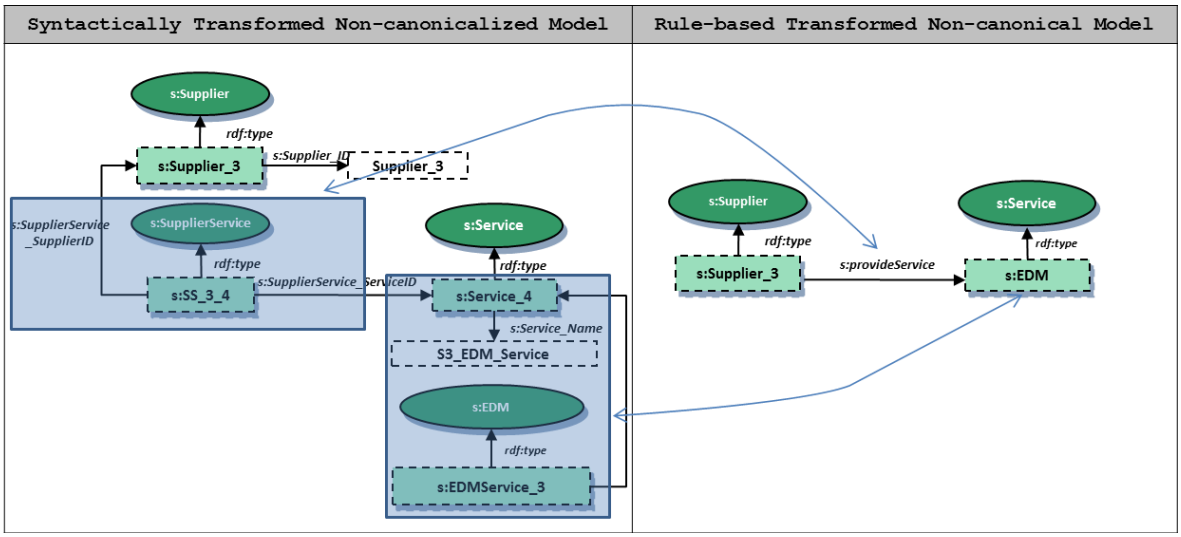


Figure 23: Additional rule-based transformation applied to the syntactically transformed MSC information

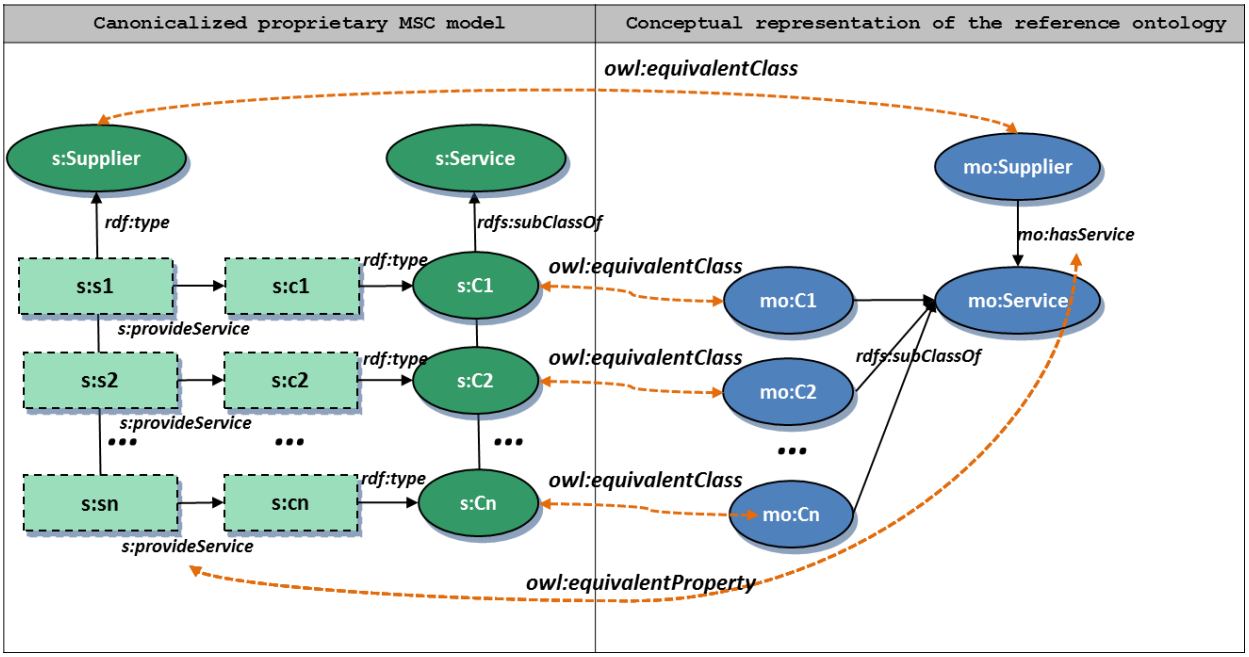


Figure 24: Quantitative analysis of OWL mapping inference after canonicalization

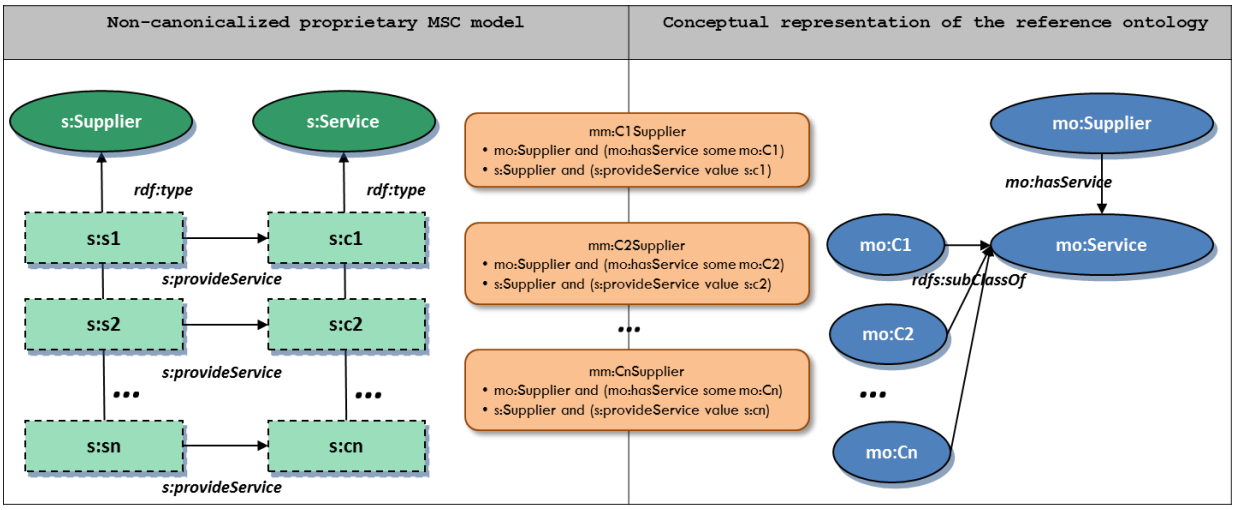


Figure 25: Quantitative analysis of OWL mapping inference without canonicalization

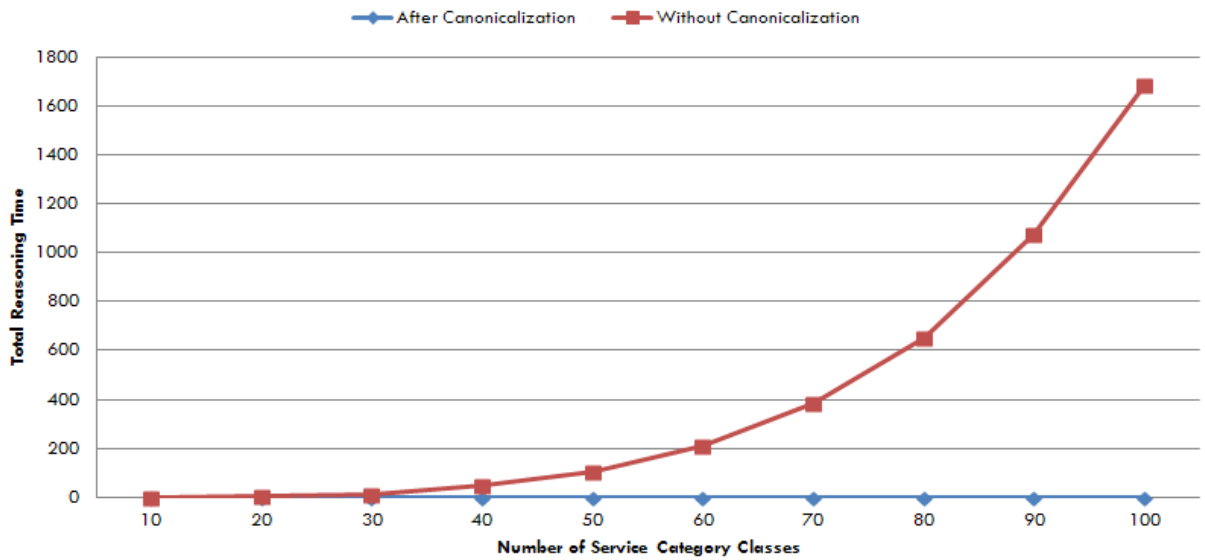


Figure 26: Aggregated reasoning performances of two cases

A Framework to Canonicalize Manufacturing Service Capability Models

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

Entity					Literal	Axiom	Binding Expression
Class	Individual	Object Property	Data Property	Data type			
C1 C2 C3 C4	I1 I2 I3	OP1 OP2	-	-	-	(I1, rdf:type, C1) (I2, rdf:type, C2) (C4, rdfs:subClassOf,C3) (I3, rdf:type, C4) (I1, OP1, I2) (I2, OP2, I3)	C1 = p:Supplier C2 = p:Profile C3 = p:ServiceCategory OP1 = p:hasProfile OP2 = p:hasService

Table 2: Initial ontology patterns correspondence

OPC ID	Source Ontology Pattern		Target Ontology Pattern
	Artifact type	Representative Artifact	
OPC1	Class	s:Supplier, s:ServiceCategory	Supplier-Service
	Individual		
	ObjectProperty		
	DataProperty		
	Datatype		
	Literal		
	Axiom		

Table 3: Ontology patterns correspondence output from the OPCs identification

OPC ID	Source Ontology Pattern		Target Ontology Pattern
	Artifact type	Representative Artifact	
OPC1	Class	s:Supplier, s:ServiceCategory	Supplier-Service
	Individual	s:SupplierA,s:EDM	
	ObjectProperty	s:hasMachiningService	
	DataProperty		
	Datatype		
	Literal		
	Axiom	(s:SupplierA, s:hasMachiningService, s:EDM)	

Table 4: Serialization of the source ontology pattern SP1 in Figure 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

Table 5: An exemplary transformation rule TR1 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

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

Table 7: Source & target ontology pattern correspondences output from the OPCs 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	
	Literal	"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	
	Literal	"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	
	Literal	"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	
	Literal	"6cm – 48cm"	
	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	
	Literal	"6cm – 48cm"	
	Axiom	(s:PartLength_4, rdf:type, s:PartLength), (s:PartLength_4, s:PartLength_Value, "6cm – 48cm")	

Table 8: Transformation rule for OPC1

Rule ID	PTR ID	Source Ontology Pattern Variable	Target Ontology Pattern Variable	Transformation type	Transformation Expression
TR1	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	-

Table 9: Transformation rule

Rule ID	PTR ID	Source Ontology Pattern	Target Ontology Pattern	Transformation type	Transformation Expression
TR5	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 information ready for the pattern transformation

OPC ID	Source Ontology Pattern	Target Ontology Pattern	Transformation Rule
OPC1	SP1	Supplier-Service	TR1
OPC2	SP2	Service-Categorization	TR2
OPC3	SP3	Service-Categorization	TR3
OPC4	SP4	Service-LengthCapability	TR4
OPC5	SP5	LengthCapability	TR5

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 TR1 application on SPI1.1 pattern instance

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	-	-

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))

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