A Survey and Classification of Principles for Domain-Specific Ontology Design Patterns Development

Marko Vujasinovic [a,b], *Nenad Ivezic* [a], *Boonserm* Kulvatunyou [a]
[a] *National Institute of Standards and Technology, Gaithersburg, MD, USA.*
[b] *INNOVA S.p.A., Rome, Italy.*
E-mails: m.vujasinovic@innova-eu.net, nenad.ivezic@nist.gov, serm@nist.gov

Abstract. Dynamic, networked service-oriented systems, like those found in manufacturing, logistics, or transportation, require efficient communication of capabilities of their services to enable on-the-fly integrations as a result of changing requirements. Previously, in a case of a manufacturing services network, we have shown the manufacturing service capability (MSC) information communication can be enhanced by introducing a reference MSC ontology – a formal, OWL DL domain-specific ontology. However, consistent and quality development of reference ontology for a large and evolving domain such as manufacturing is a challenge. Therefore, we propose to utilize the notion of OWL ontology design patterns (ODPs) to develop such reference ontology. However, despite the existence of rich design patterns for information modeling in general, there has been no documentation detailing the principles for development of domain-specific ODPs for domain-specific semantic models. This survey paper fills this void by providing a survey and systematic synthesis of applicable principles for domain-specific ODP development in an investigation of the prior works in data modeling, object-oriented software analysis, and ontology modeling design patterns. The paper discusses applicability of the revealed principles in regards to requirements of MSC domain-specific ODPs. Although the paper is concerned with the MSC domain, the findings apply to any domain-specific ODP development. Further research is identified to operationalize principles towards domain-specific ODP development.

1 Introduction

Dynamic, networked service-oriented systems, including those found in manufacturing, logistics, or transportation, increasingly require efficient communication of their capabilities and on-the-fly integrations as a result of changing requirements. For example, dynamic production networks depend on the efficiency of supply chain assemblies and reconfigurations, which, in turn, depend on the efficient communication of manufacturing service capabilities (MSC) and matchmaking between manufacturing customers' needs and suppliers' capabilities. MSC information communication and matchmaking includes information about both quantitative and qualitative properties of the services, such as the ability or capacity of production processes and resources, expertise, certifications, or digital information processing ability.

Presently, the communication of MSC information happens on the Web and is based on various proprietary and syntactic MSC vocabularies and folksonomies. The matchmaking between available capabilities of manufacturing service providers and requirements of manufacturing service customers is syntactic- and text-based, and is very limited because of the semantic ambiguities, poor expressivity, and cross-integration issues of various proprietary MSC information models and vocabularies (Kulvatunyou, Ivezic, & Lee, 2013).

Semantic enrichment using a reference ontology[1] is an approach that can help address these limitations. In our prior work (Kulvatunyou, Ivezic, & Lee, 2013), we explored semantic enrichment of MSC information using an experimental reference MSC ontology represented in OWL DL (McGuinness & Van Harmelen, 2004) - an ontology language with formal semantics based on description logic. In particular, we developed a prototype for OWL DL-based semantic integration of different proprietary MSC models using a reference ontology, and semantic querying using the different proprietary MSC terminologies. OWL DL, as a language of our choice, provided the expressivity needed for conceptualization and semantic description of presently syntactic MSC information. DL reasoners provided for consistency checking, classification, and semantic interpretation of MSC information. The potential of OWL DL is also evident from other research related to the MSC domain (Ameri & Dutta, 2006), (Jang, Jeong, Kulvatunyou, Chang, & Cho, 2008) and (Im, Lee, Kim, Peng, & Cho, 2010).

While OWL DL can provide these benefits, developing a consistent and quality reference ontology in OWL DL remains a challenge, especially for a large and dynamically evolving domain such as manufacturing. However, use of OWL ontology design patterns (ODPs, in short), tailored to specific domain requirements, carries the promise to significantly ease ontology development, evolution, and mapping (Kulvatunyou, Lee, & Ivezic, 2013). We expect

[1] A reference ontology is a shared, formal and computer-interpretable specification of a particular domain conceptualizations (Uschold & Gruninger, 1996).

such ODPs to take form of reusable templates - structural and parameterized forms that modelers instantiate and populate with terms from domain-specific vocabulary. The ODPs would allow capture of recurring contents and structures of the domain information models in a uniform and consistent way.

Design patterns have emerged as engineering artifacts in many engineering disciplines, most notably in software engineering. According to Fowler (1997), "a pattern is an idea that has been useful in one practical context and will probably be useful in others." Or, according to Gamma et al. (1995), "a design pattern systematically names, motivates, and explains a general design that addresses a recurring design problem ... it describes the problem, the solution, when to apply the solution, and its consequences." Because of the enormous success of design patterns in software engineering, researchers began using the idea for ontology engineering and introduced ontology design patterns (ODPs) as "best practice" modeling solution in OWL (W3C, 2005; Aranguren, 2008). ODPs can capture foundational and core domain concepts to provide reusable solutions "for solving design problems for the domain classes and properties that populate an ontology" (Gangemi, 2005).

However, despite the existence of rich design patterns, there has been no clear documentation detailing the principles for development of domain-specific design patterns that are tailored to requirements of a specific domain. So far, the efforts of researcher and engineers have been focused on synthesizing collections of design patterns, not on synthesizing and structuring the design pattern development principles. Hence, this survey paper provides a systematic synthesis of applicable principles for domain-specific ODP development and, in addition, discusses the applicability of these principles from the perspective of MSC domain-specific requirements. We survey relevant results across related work areas, including data modeling, object-oriented software analysis information models, and ontology-modeling design patterns[2].

The paper is organized as follows. Section 2 discusses notable semantic modeling challenges that may be mitigated by using domain-specific ODPs. The perspective of the discussion is MSC information modeling specific, but the challenges apply to other domains, too. Then, Section 3 clarifies a notion of ODP and provides a summary of requirements of ODPs for domain-specific information using the MSC information as an example. Section 4 then discusses the state of the art – in particular it provides a survey into relevant results across related work areas, including data modeling, object-oriented software analysis information models, and ontology-modeling design patterns, to derive principles for information modeling design patterns development. Here, we introduce a methodological grid based on four dimensions to analyze and discuss the state of the art. The dimensions are: (1) typology, (2) vertical-horizontal variability, (3) design principles, and (4) representation of design patterns. Section 5 synthesizes the identified and derived principles to design-pattern development and outlines general principles for development of ODPs for the domain-specific information. Finally, Section 6 provides conclusion and future research directions.

2 Challenges in a domain-specific ontology modeling

As noted, our approach is to develop a reference MSC ontology where MSC terms and their semantics are formally encoded using OWL - whether as OWL classes, properties, or individuals. Let's discuss some of the challenges and related issues that ontology developers may encounter during the semantic modeling of domain-specific information.

First of all, the ontology developer may use alternative solutions for capturing domain knowledge in an ontology using OWL DL. A very simple example of that would be modeling of descriptive features (or, qualities, attributes or modifiers) of things captured as ontology concepts (Rector, 2005). There are at least two approaches for modeling the descriptive features: (#1) as individuals whose enumeration make up the parent class representing the feature; (#2) as disjoint classes which exhaustively partition the parent class representing the feature (Rector, 2005). However, using both approaches to address essentially the same kind of requirement in a particular ontology may end-up in inconsistent design throughout the ontology. The ontology that has the inconsistent design may be problematic for the further evolution, refactoring, and even querying.

Then, the ontology developer may encounter insufficient expressiveness of the OWL DL to naturally capture certain requirements, which then would require workarounds, if even possible. For instance, in OWL, a property is always a binary relation. It links two individuals or an individual and a value. However, there could be cases where an n-ary relation, which links an individual to more than just one individual or value, is more appropriate. N-ary relations are

[2] Design patterns, such as (Gamma, Helm, Ralph, & Vlissides, 1995) patterns, that are concerned with architectural, operational and implementation aspects of software systems are not of interest to this research. Only works in design patterns relevant to information modeling are taken into analysis.

not natively supported in OWL DL; however, there are alternative workarounds such as ones proposed in (Hayes & Welty, 2006).

Further, the ontology developer doing mapping between different ontologies can produce complex and less manageable mappings, unless ontology design is uniform and consistent within and across the ontologies. For instance, if the above mentioned "features" are modeled using approach #1, then mappings will refer to instances (e.g., by `owl:sameAs` or `rdf:type` constructs). On the other hand, if the "features" are modeled using approach #2, then mappings will refer to classes (e.g., by `rdf:type` or `owl:equivalentClass` or `owl:subclassOf` constructs). Therefore, essentially the same mapping situations would involve mappings that are established differently because of differences in the design of ontologies.

In addition, development of a reference ontology for a large and dynamically evolving information domain is not a one-time process. Such reference ontology continues to evolve as new requirements and new knowledge emerge. Therefore, when it comes to the ontology evolution, not carefully managed changes in an ontology may produce hard-to-debug inconsistencies in already captured knowledge, invalid ontology mappings, or a need for a complete remapping. What is needed is an ontology engineering approach where concepts and their relationships are uniformly and consistently designed as ontology requirements evolve over long periods of time.

3 Use of domain-specific ontology design patterns (ODPs)

3.1 An example of a domain-specific ODP

Applying different approaches for modeling essentially the same situation in a specific context and domain may result in heterogeneous domain-specific ontologies that can become unsuitable for mapping, querying, debugging, and maintaining. One possible solution to mitigate the issues is to establish an evolving library of domain-specific ODPs, which would be used and strictly adhered to in the reference ontology development. When the same ODPs are applied to different ontologies, the ontologies become more similar in terms of design, comparison, and mapping.

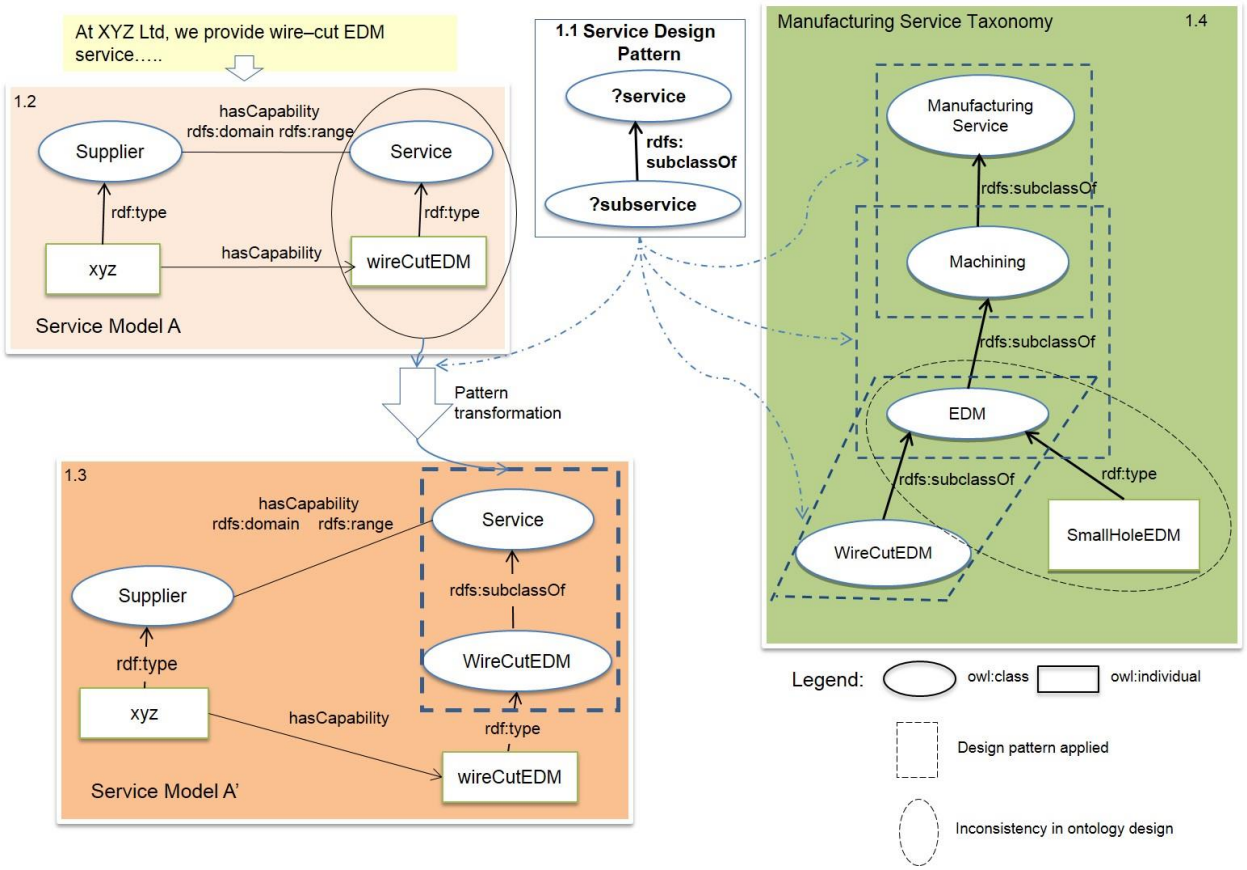
Figure 1 illustrates the idea of domain-specific ODPs on an example of one hypothetical MSC model and Manufacturing Service taxonomy, which is a backbone of MSC reference ontology. In particular, in rectangle 1.1, the figure shows a Service Design Pattern[3] - one of many ODPs in a library - for modeling a taxonomy of manufacturing services. The Service Design Pattern "says" that each particular manufacturing service e.g. Wire Cut EDM (Electric Discharge Machining[4]) or Small Hole EDM, should be a subclass of a more general service concept; i.e., they should be arranged into a taxonomy. This design patterns has `?service` and `?subservice` placeholders that need to be replaced with OWL classes representing actual manufacturing services e.g. `owl:ManufacturingService` and `owl:Machining` , or `owl:Machining and owl:EDM`, as illustrated in rectangle 1.4 of the figure. Assume now Model A (rectangle 1.2) is a description of manufacturing services provided by XYZ Ltd supplier. `'xyz'` is a Supplier that provides a `wireCutEDM` service. Wire Cut EDM is a special type of EDM service. Model A, however, does not conform to the Service Design Pattern, since `wireCutEDM` is directly an instance of a `Service` class and there is no subclass that represents the specific manufacturing service (in this case, subclass representing a Wire Cut EDM). To become compliant to the design proposed, Model A should be transformed into Model A' (Rectangle 1.3). In (Kulvatunyou, Lee, & Ivezic, 2013), the authors provided a methodology for transforming manufacturing service models into ones that conform to the given ODPs. In Model A', `WireCutEDM` class is introduced and it is a subclass of `Service`, while `wireCutEDM` has become an instance of the `WireCutEDM` class.

The manufacturing service taxonomy (Rectangle 1.4 of Figure 1) illustrates the case of three conformant examples of the ODP application and one non-conformant example (`SmallHoleEDM`). For example, to make this model conformant, `SmallHoleEDM` must be captured as a subclass of the `EDM` concept.

[3] Disclaimer: Service Design Pattern is a hypothetical ODP, for illustration purpose only. It is not proposed here as the best modeling practice to model taxonomy of manufacturing services, but rather to illustrate the idea of domain-specific ODP library and its application for semantic modeling of domain-specific information.

[4] A manufacturing process whereby a desired shape is obtained using electrical discharges (sparks) Source: en.wikipedia.org/wiki/Electrical_discharge_machining

Figure 1 Application of an ODP in semantic modeling of MSC information



Currently available ODPs are those published at the ODP portal (http://ontologydesignpatterns.org/), which was established under NeOn European FP7 project (Presutti, et al., 2008), then ODPs in the Manchester (2009) and W3C (2005) libraries. However, those currently available ODPs are not complete references for designing domain-specific ontologies, and not necessarily adaptable to specific requirements of domain-specific information communication such as MSC information communication.

Obviously, domain-specific ODPs need to be created before the domain-specific ontology development begins. The domain-specific ODPs then may evolve and new ODPs may emerge as the domain expands and the new knowledge to be captured in the ontology emerges. To establish domain-specific ODPs two things are essential: requirements that ODPs have to satisfy and design principles for their development. Below, we provide general requirements that the MSC ODPs have to satisfy, while in Section 5 we discuss possible principles for the development of MSC ODPs for the requirements.

3.2 Requirements of domain-specific ODPs

Requirements of domain-specific ODPs may come from three different but complementary sources: the reference ontology requirements (source S1), the information communication requirements (source S2), and the required characteristics of domain-specific ODPs (source S3).

In particular, ontology requirements can be functional or nonfunctional (Gomez-Perez, Fernandez-Lopez, & Corcho, 2003) and ODP requirements coming from the reference ontology can be categorized in a similar way, either as functional or nonfunctional. The functional requirements are intended, content-related uses of the ontology (or, of an ontology design pattern) such as retrieval, inference and validation of domain knowledge or domain information. The nonfunctional requirements are characteristics of the ontology (or, of an ontology design pattern) such as computational efficiency, clarity, reusability, etc.

The functional requirements, whether in case of an ontology or an ODP, can be expressed as competency questions (CQs, for short)[5], while the nonfunctional requirements need other means of expression. In particular, CQs may determine a conceptual coverage of an ontology or of an ODP. The required conceptual coverage determines what concepts will be captured, their scope, and structural or logical relationships. Let's discuss below the requirements of MSC ODPs, according to their sources. A summary of MSC ODP requirements, according to their origin and type, is given in Table 1.

Requirements' source 1 - functional: The functional requirements coming from the reference MSC ontology are retrieval, inference, and validation requirements. These requirements may be related either to MSC concepts in the reference ontology, actual MSC descriptions in a model that is linked to the reference ontology, or both. An information-retrieval CQ related to actual MSC descriptions is, "Which suppliers have a Wire EDM Process with capability up to 30-degree taper cuts?" Whereas, an information- retrieval CQ related to a MSC concept would be, "What is the maximum degree taper cut in a wire EDM Process?" Information-retrieval CQs often require inference over the MSC information. For example, "Which suppliers have an EDM Service?" *implicitly* requires that suppliers having different kinds of EDM services be included in the answer. In (Suarez-Figueroa M., Gomez-Perez, Motta, & Gangemi, 2012) the authors actually propose explicit specification of reasoning (inference) requirements. Importantly to note is that their notion of competency question requirements is very similar to our notion of the retrieval requirements, if not the same. For example "Which suppliers have an EDM Service and specializations of EDM Service" and "What are the generalization and specializations of a given manufacturing service?" are CQs that more explicitly specify the inference requirement. Finally, CQs oriented to validation ask if MSC ontology contains sufficient and valid axioms to ensure correct and complete definitions of MSC information. Validation requirements can be contextual statements as proposed in (Suarez-Figueroa M.,Gomez-Perez, Motta, & Gangemi, 2012). Figure 2 presents an illustrative set of CQs for a reference MSC ontology[6].

Requirements' source 1 - nonfunctional: The nonfunctional requirements coming from the reference MSC ontology include the requirement of uniformity and consistency in MSC ontology design, as pointed out before. This requirement could be described more specifically as a requirement to consistently use established ODPs throughout the reference MSC ontology. Another nonfunctional requirement associated to reference MSC ontology is a computational efficiency on ontology reasoning tasks.

Requirements' source 2: Successful MSC information communication requires a mapping between proprietary MSC models and the reference MSC ontology. Such mappings achieve reconciliation of naming, structural, and semantic differences in a heterogeneous MSC information sharing environment. An important requirement in the reconciliation is to have computationally efficient and straightforward ontology mappings. A requirement for computational efficiency of ontology reasoning tasks such as MSC information retrieval/inference is also an MSC information communication requirement.

Requirements' source 3: MSC ODPs have to be reusable in different but essentially the same modeling situation. They have to also be understandable to end-users who use ODPs for modeling, such as ontology developers.

Table 1 MSC ODP requirements

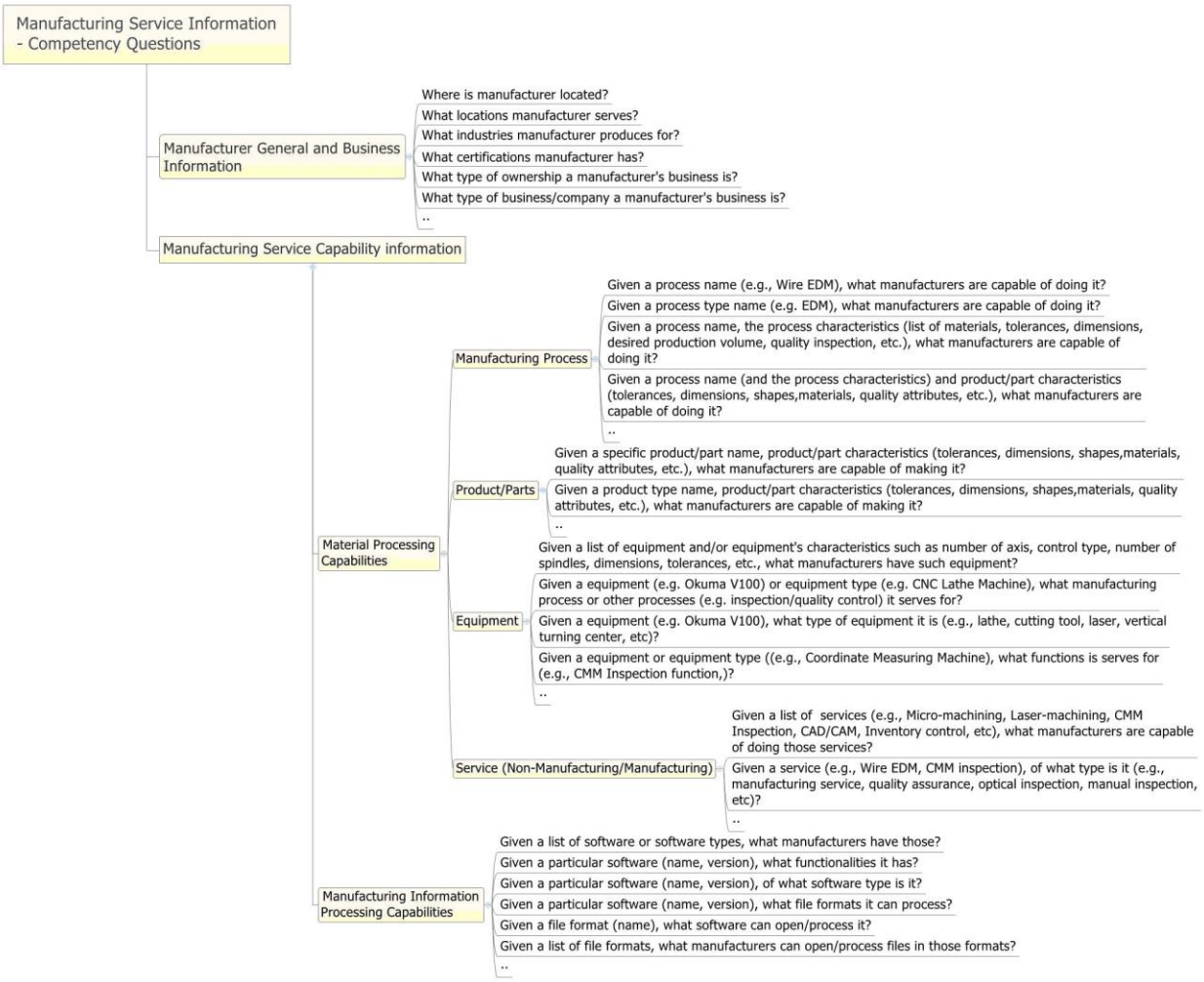
No.	Origin	Requirement	Type
R1	S1	MSC ODP enables consistent representation of portions of MSC information with common abstraction to support MSC information retrieval	Functional
R2	S1	MSC ODP enables consistent inference on portions of MSC information with common abstraction	Functional
R3	S1	MSC ODP enables consistent validation on portions of MSC information with common abstraction	Functional
R4	S1	MSC ODP provides for uniform ontology design of MSC information	Nonfunctional

[5] CQ technique (Gruninger & Fox, 1994) is a way to specify ontology information retrieval requirements as natural or machine-language questions that the ontology must be able to answer. Ontology design pattern-level functional requirements can be specified using the CQ technique as well (Suarez-Figueroa M.,Gomez-Perez, Motta, & Gangemi, 2012). While some researchers refer to CQs as only questions about concepts (De Nicola, Missikoff, & Navigli, 2009), CQs can be about any aspect of ontology (or, ontology design pattern) content including ontology instances.

[6] It is important to note that there is not an a priori relationship between CQs and design patterns. That is, one or more design pattern(s) might be needed to enable a single CQ. For example, there could be one MSC ontology design pattern for capturing a business party such as supplier, and another one for capturing a manufacturing service such as an EDM service. Then, both these ontology design patterns may be needed to support "Which suppliers have an EDM capability?" CQ. If CQ is "What type of ownership a manufacturer's business is?" only the MSC ontology design pattern for capturing a business party might be needed. Discussion of how to identify all needed MSC ontology design patterns is out of scope of this paper.

R.5	S1, S2	MSC ODP meets computational efficiency requirement on MSC information retrieval and inference	Nonfunctional
R6	S2	MSC ODP enables computationally efficient and straightforward MSC ontology mappings	Nonfunctional
R7	S3	MSC ODPs is formally representable to support consistent reuse (i.e. reusability)	Nonfunctional
R8	S3	MSC ODP is clear, understandable (i.e., clarity)	Nonfunctional

Figure 2 CQs for MSC information. The CQs are grouped around central information concepts in MSC domain (illustrated using a rectangle)



4 A survey of works in design patterns and principles to their development

This section is a survey of existing works in design patterns and principles to their development. In particular, the section discusses (1) typology, (2) variability, (3) design principles, and (4) representation of design patterns for a variety of information modeling approaches. The typology refers to a classification of design patterns according to their application - whether they are solutions to conceptual, logical or structural models, or just a syntactic sugar. The variability can be vertical or horizontal, as we will explain in Subsection 4.1.2. Design principles refer to any principle used or proposed for developing design patterns or for identification of design patterns. The representation refers to the current means for documenting design patterns and making them available for end-users. In addition to these four dimensions, a brief discussion of the applicability of existing design patterns to semantic modeling of domain-specific information is given.

4.1 Data modeling patterns

Data modeling patterns are reusable data models that can be used "to develop high-quality data models in short amounts of time" (Silverston & Agnew, 2011). They are "profound, recurring modeling fragments that provide a proven solution for some modeling problem" (Blaha, 2010). Note, a data model here means the entity-relationship model introduced in (Chen, 1976). The data modeling patterns considered here are from (Hay, 1996), (Silverston, 2001a), (Silverston, 2001b), (Silverston & Agnew, 2011), and (Blaha, 2010), which we have found to be the most relevant in this field of our interest.

4.1.1 Typology of data modeling patterns

We found three different types of data modeling patterns in the considered literature - logical patterns, physical patterns, and structural patterns.

Logical patterns are reusable models that capture domain entities and their relationships, devoid of their physical implementation details. They are called logical because they are patterns for logical data models. On the other side, physical patterns are implementations of logical patterns in a target platform technology; they are parts of physical data model. For one logical pattern, there could be many different corresponding physical patterns depending on target platform, data manipulation, and access strategies. For instance, Hay's patterns are all pure logical patterns, with no concerns of physical implementations, while Silverston's and Blaha's patterns encompass both logical and physical aspects. We will mention some of their patterns in the following section (4.1.2).

Structural patterns are not concerned with capturing entities' content, as opposed to the logical and their physical patterns; but rather with the structural organization of the entities. Structural patterns, as those in (Silverston & Agnew, 2011), are patterns for creating hierarchical relationships, aggregation (sets) of entities, or categories and classification of entities, and graphs. They have a basis in topology and graph theory (Blaha, 2010). For example, Blaha's '*Hardcoded Tree'* pattern is a structural pattern for capturing organization of hierarchies whose levels and type sequences do not change frequently over time. Or, the '*Directed Graph'* pattern for capturing structures such as map of flights between airports, or a collection of equipment in the manufacturing plant (Blaha, 2010). Hay (1996) also provides structural patterns to model hierarchies, classification or categories of entities. The structural patterns can be logical, devoid of actual implementation details, or physical, addressing actual implementation details in a target platform.

4.1.2 Vertical-horizontal variability in data modeling patterns

Based on the terminology in (Verelst, 2004), we define horizontal and vertical variability of design patterns as follows. Horizontal variability is the possibility of organizing the design patterns into two general groups: universally applicable (or, cross-domain) and domain-specific (e.g. for manufacturing information modeling or health-care information modeling), without having a specialization relation between the universally applicable and domain-specific ones. On another side, the vertical variability is further specialization of the universally-applicable or domain-specific design patterns. E.g. the design patterns for manufacturing information modeling may be specialized into the discrete manufacturing and process manufacturing information modeling patterns. Therefore, the design patterns vertically laid are specializations of the design patterns horizontally laid and, in a more formal way, could be linked to them using the subsumption (is-a) relationship. A domain-specific design pattern may or may not be a vertical variation of a universally-applicable design pattern.

For example, in (Hay, 1996), universal data modeling patterns are *Parties*, *Organizations*, or *Products;* while *Structure and Fluid Path*, *Flow*, or *Process* are not specializations of the universal patterns but rather specific patterns for process manufacturing industry. Those patterns are horizontally laid. Many of Hay's data modeling patterns capturing universal concepts (e.g. *Party)*, can be specialized into subtypes (e.g., *Employee*), which can be viewed as their vertical variability and patterns at a lower level of generalization.

Similarly to Hay's, Silverston's patterns also span from universal to domain-specific. Silverston's universal data modeling patterns are similar to Hay's (e.g., *Parties*, *Products*, *Parts*, or *Product Category).* Silverston (2001b) provides domain-specific data modeling patterns for data models for discrete manufacturing. For instance, *Part Substitution*, *Inventory Item Configuration*, *Production Runs* or *Production Run Types*. Silverston also provides for capturing both universal concepts (e.g. *Party)* and their vertical specializations (e.g., *ShipToParty*, *ShipFromParty* specialization of *Party*). In (Silverston & Agnew, 2011), the authors provided highly generalized data modeling patterns, such as *Declarative Role*, or *Contextual Role*, which provide for modeling of any entity roles.

Blaha's (2010) patterns are also at different levels of generalization. For example, his *Item-ItemDescription* is a pattern highly generalized to provide models that capture data of some *item* and that *item description*, regardless of

what the item is. For example, Aircraft with 'tailNumber' attribute, is an item, and AircraftDescription with attributes 'manufacturer' and 'models', is the item description. Blaha's patterns, which he calls archetypes, are all universal and generalized data concepts. The archetypes may be applied or specialized for particular domains. For instance, the archetypes are *Actor*, *Address*, *Position*, *Product*, or *Part*. Additionally to the archetypes, Blaha introduces "canonical model patterns" as ready-to-use complete data models for specific purposes.

4.1.3 Design principles for data modeling patterns

The preceding data modeling patterns are empirically grounded, which means derived from the accumulated practical experience in resolving recurring data modeling tasks. Their design and development principles are not clearly documented; however, by analyzing their documentation, we derived several pattern design principles, as discussed next.

A *model size principle (DP1)*[7] that states how many artifacts, such as entities or relationships, should be incorporated into a data modeling pattern. For instance, Blaha more specifically says that a data modeling pattern should encompass more than a single, isolated entity but less than ten entities and relationships *(DP1.1)*. Hay's principle is to have the "purest" data model possible *(DP1.2)*. This puts an emphasis on describing things in terms that are abstract enough to encompass a wide range of circumstances *(DP.1.3)* and provide a starting point, rather than a complete solution *(DP1.4)*.

A *normalization principle (DP2)* that asks for certain normalization form of data modeling patterns. One of the earliest such normalization forms, the Third Normal Form (3NF), was developed in (Codd, 1972) for relational databases' schemas. 3NF principle is very specific to the relation database modeling and its application ensures referential integrity in the database and no data duplication. In short, 3NF suggest that the non-primary attributes of a database table should be dependent on the primary key only, and that every non-primary attribute of the table should be dependent on the whole of a primary key when the primary key consists of two or more attributes. In (Silverston, 2001a), the author argued that data modeling patterns should also be developed with the normalization principle in mind, by applying 3NF normalization. In this same paper, the author describes an important related principle, called a *no-derive attribute* principle *(DP3)*. This associated principle keeps data model patterns devoid of attributes that can be derived from other attributes. Importantly, the normalization does not necessarily apply to physical data model patterns since these patterns may be de-normalized to meet performances in accessing and updating the data. Logical data model patterns are typically normalized into 3NF.

A *generalization principle (DP4)* that seems to be a prominent design principle for generalized data modeling patterns in (Silverston & Agnew, 2011). This principle recognizes that wide variety of information requirements can be captured in the same data structure. Sometimes this principle is called an *abstraction (DP5)*. Specifically, Blaha (2010) pointed out an applicability and abstraction as characteristic of his archetypes. The archetypes are universally applicable concepts, not domain-specific concepts. Certainly, the wider applicability implies higher abstraction in data modeling patterns.

The DP1-DP5 principles address mainly the functional aspects of design patterns, and so, determine the conceptual coverage of a data modeling pattern.

Further, Blaha highlights several anti-patterns that should be avoided in data modeling for relational databases. Those anti-patterns include symmetric relationships, ambiguous naming and descriptions of entities, disconnected-isolated entities/properties, or multiple inheritances. Blaha's *avoidance of anti-patterns principle (DP6)* can be seen as a general principle where each specific anti-pattern recommendation is a specific principle in patterns design.

On the other side, nonfunctional requirements are determined with operational aspects such as physical implementation concerns, computational performance, or maintainability of data modeling patterns. Depending on specific nonfunctional requirements specific we found out that techniques such as indexing policies, de-normalization of tables, or table partitioning strategies, are typically applied. For physical data model patterns, the invariability of a database schema to additional, new, logical entities may be a requirement, which can be met by a generic table structure for storage of a variety of entities. So, such observation may be interpreted also as the generalization principles. The intended purpose of data can determine the design of patterns. For example, if data are going to be used for analytical purposes, physical data model patterns should provide for that purpose. An example of such a pattern is a STAR schema (Silverston, 2001a) that binds data to different analytical dimensions. This may be a design principle formulated as - *adhere to a structure designed to meet required operational aspects (DP7)*.

[7] DP stands for a derived principle. We assign to each derived design principle an unique id to enhance their traceability between this and next section of the paper.

4.1.4 Representation of data modeling patterns

Data modeling patterns are frequently represented using graphical diagrams and textual descriptions. Hay and Silverston used Case*Method (Barker, 1990) data modeling notation. Blaha used both UML (Unified Modeling Language) and IDEF1X (Integration Definition for Information Modeling) notations. Additionally to the graphical representation, Silverston provided computer-processable representations of his patterns via SQL scripts to allow their use and adoption for building the data or database models. For example, one can use SQL representation of data modeling patterns and utilize them by importing scripts into CASE tool for adoption and further refinement. Case*Method, UML or IDEF1X notation suffice for logical data model patterns. However, SQL representation is necessary for physical data model patterns. That is because UML and IDEF1X to SQL mapping might not be one-to-one. Hence, additional implementation concerns need to be considered when using data modeling patterns captured in UML or IDEF1X.

4.1.5 Applicability to semantic modeling of the domain-specific information

Data modeling patterns are design patterns provided primarily for design of E-R (relational database) models. Those patterns contain details such as primary keys, foreign keys, alternative keys, and associative entities for capturing many-to-many relationships between entities. Having such details, it is questionable if those design patterns can provide for reference ontology development, even if available in OWL representation. Relational models and OWL ontologies are significantly different since OWL ontologies can capture both the content and semantics of entities.

Assuming the data modeling patterns translated into OWL, they could serve as some starting, but likely distant, point in capturing semantics of domain-specific information. For instance, Hay's *Geographic Location* pattern encompassing party and location entities, if implemented in OWL, may provide for 'Where is manufacturer located?' or 'What locations a manufacturer serves?', in a case of MSC domain. Composition of Blaha's *Actor* and *Location* patterns could provide for that purpose as well. Terminologically, there are many matching concepts in the data modeling patterns to concepts in MSC domain. For instance, Part, Product, Product Category, Location, Unit of Measure, Measure, Process, Condition, Asset/Equipment Type, Material, or Material Type. But many concepts might be missing.

Certainly, additional effort would be needed to re-design the data modeling patterns from relational database viewpoint to OWL DL perspective. In particular, Blomqvist (2010) has developed several ODPs by translating some of Hay's and Silverston's patterns into OWL DL. A one-to-one translation approach was applied where an entity/property in a data model pattern was mapped to a concept/property in OWL. Then, patterns were reviewed with help of domain experts and updated where necessary since the patterns inherently assumed a data modeling paradigm not easily translatable into the DL paradigm (Blomqvist, 2010). Nevertheless, even if potentially applicable data modeling patterns are provided in OWL DL form, their further adaptation (e.g., specialization, extension, simplification) would be needed to meet the domain-specific information requirements.

4.2 Software analysis model patterns

In general, software development patterns are a means to enhance software quality, flexibility and maintainability, and to reduce development time. Different types of software development patterns are proposed for different phases of software development process. Software analysis patterns support the software analysis phase that is concerned with development of application-specific conceptual models such as object-oriented domain models. Software design patterns, such as (Gamma, Helm, Ralph, & Vlissides, 1995) design patterns, support the software design phase that is concerned with architectural, operational and implementation aspects of software systems. Implementation patterns, as particular programming language idioms, support the low-level software coding. In particular, our interest is in software analysis (object-oriented) model patterns. Software analysis patterns, or analysis model patterns, are "groups of concepts that represent a common structure in modeling" (Fowler, 1997) that can be used to build software analysis models, which are in fact information models, but object-oriented. Software design patterns and software implementation patterns are not applicable for information or conceptual modeling, hence, excluded from the further discussion. Software analysis model patterns taken here into the discussion are from (Coad, 1992), (Fowler, 1997) and (Arlow & Neustadt, 2004), which we have found to be the most representative in this field of our interest.

4.2.1 Typology of software analysis model patterns

The analysis patterns proposed in (Coad, 1992), (Fowler, 1997) and (Arlow & Neustadt, 2004) are of similar nature to the data modeling patterns, which we discussed in Section 4.1, in the sense that they provide also for the essentially the same purpose, which is information or conceptual modeling. Therefore, we identified conceptual

analysis model patterns and structural analysis model patterns as two different types of software analysis model patterns.

Specifically, conceptual analysis model patterns, such as Fowler's or Arlow's *Party*, or Coad's *Item-ItemDescription* pattern, provide for capturing business concepts and their properties. Structural analysis model patterns provide for structural organization of data, such as hierarchical organization, or n-ary relationships (e.g. Fowler's *Organization Hierarchy* pattern, Arlows' *N-ary relationship*).

Conceptual analysis model patterns and structural analysis model patterns are of similar purpose and scope to the logical data model patterns and structural data modeling patterns, respectively. However, they differ in a modeling paradigm. Data modeling patterns are influenced by the relational modeling paradigm, while software analysis patterns are influenced by the object-oriented modeling paradigm.

We note that Arlow & Neustadt (2004) have introduced archetypes and archetype patterns as software analysis patterns. According to Arlow & Neustadt, an archetype is defined as "a primordial thing or circumstance that recurs consistently and is thought to be a single universal concept." An archetype pattern is explained as collaboration between archetypes that occurs consistently and universally in business contexts. For instance, their *Party* archetype pattern is a composition of several archetypes including *Party*, *Organization* and *Address* archetypes. An archetype pattern notion is essentially the same as software analysis pattern notion, with the difference that archetype patterns are always concerned with Arlow & Neustadt's archetypical concepts.

4.2.2 Vertical-horizontal variability in software analysis model patterns

Similarly to data modeling patterns, software analysis patterns may vary vertically and horizontally. For instance, Fowler's (1997) analysis patterns are organized by domains where patterns emerged from domains such as Accountability, Observation and Measurements, Inventory and Accounting, Planning, and Trading. This organization does not imply that all the Fowler's patterns are domain-specific. Some are universally applicable (e.g., *Party, Organization, Observation, Measurement, Atomic Unit, Compound Unit*), while others are applicable to particular domains only (e.g., *Accounts*, *Transactions* or *Balance Sheet* for an accounting domain). Fowler provides a number of specializations of his universally-applicable patterns. Examples include observations and measurements specialized for corporate finances.

Archetypes and archetype patterns in (Arlow & Neustadt, 2004) capture universal business concepts such as *Party*, *Organization*, *Address*, *Money*, *Order*, *Product*, *Communication*, *Responsibility*, *or Product Catalog* archetypes/patterns. There are domain-specific archetype patterns such as *Customer Relationship Management* archetype pattern. According to Arlow & Neustadt (2004), their archetypes and archetype patterns may be specialized or extended to adapt to specific context. For example, *EmailAddress* or *GeographicAddress* archetypes are specializations of *Address* archetype. Hence, the vertical-horizontals variability is very often present in the collection of software analysis patterns.

Notably, software analysis patterns are often abstractions. As such, they can capture abstract concepts that have no real-world counterparts. For instance, Coad's *Item-ItemDescription* analysis pattern is an abstraction of things that share the same description (metadata). Structural software analysis patterns are generally abstractions of structural concepts.

4.2.3 Design principles for software analysis model patterns

To the best of our knowledge, the analyzed object-oriented software analysis model patterns are also empirically grounded. That is, they are not provided within a structured methodology having an explicit set of pattern design principles. Similarly to data modeling patterns, conceptual coverage of software analysis model patterns reveals supported functional requirements. Software analysis patterns in (Coad, 1992), (Fowler, 1997), and (Arlow & Neustadt, 2004) seem unconcerned with run-time and implementation requirements such as operational performance or memory storage requirements, which are of great concern to data modeling patterns.

In particular, Coad (1992) does not provide design principles for his patterns explicitly. Rather, he emphasizes the reoccurrence as the main characteristic of his patterns. Coad points out that a pattern is a recurring structure of classes and objects that applies again and again in different object-oriented analysis and design efforts. Coad's observation is that patterns in object-oriented analysis may be found by identifying recurring structures in software analysis models and observing those lowest-level building components and their relationships to establish higher-level components for their further reusability in modeling. The reoccurrence is more a pattern identification principle, than a pattern design principle. However, to us, it could be considered as a design principle if paraphrased into a rule saying that patterns should *encompass only the concepts that reoccur in a particular domain or across domains and attributes of those concepts as well as relationships between the concepts (DP8)*.

Next, according to Fowler, the software analysis model patterns are discovered "by looking at what happens in day-to-day development, rather than by academic invention." However, from Fowler's work we derived two design principles. He suggests that software analysis model patterns should be *the simplest model possible, devoid of any flexibility if flexibility is unlikely to be utilized (DP9)*. This principle can be called as a minimal conceptual coverage or *minimal model size (same as the DP1)*. In particular to the model size, all Fowler's analysis patterns encompass *more than single concept, but no more than a few concepts (DP10)*. Another of Fowler's suggestions is a *sufficient abstraction* of a pattern *(DP11)*, to provide for applicability of patterns in different contexts. These two Fowler principles come as no surprise as aimed to patterns which serve as only a starting point in modeling.

In (Arlow & Neustadt, 2004), the authors argued that there are four essential characteristics of the business archetypes and archetype patterns as software analysis model patterns: (1) universality - consistent occurrence in business domains and systems, (2) pervasiveness – occurrence in both the business and software domain, (3) deep history - a long time existence, and (4) self-evidence to domain experts. For us, these four characteristics reveal a design principle saying particularly that patterns should *capture universal, recurring concept in some domain (DP12)*. The universal and recurring concepts in some domain are naturally self-evident to the domain experts, so we don't consider the 'self-evidence' as an additional principle, but as intrinsic to the DP12 principle. The pervasiveness, to us, should be contemplated as the pattern design principle. Further, Arlow and Neustadt emphasize the *principle of variation (DP13)* in archetypes. This means that archetypes have invariant and variant parts, which allow them to achieve applicability in different contexts that may require different models of the same thing. Arlow and Neustadt incorporated the convergent engineering approach (Taylor, 1995) in creation of archetype patterns. This approach proposes business modeling by means of object-oriented modeling to ease conversion of business models into object-oriented software systems. In the convergent engineering, business modeling begins with the identification of business elements and their subclasses and ends by establishing relationships between business elements to capture how they work together. According to Arlow and Neustadt, archetypes and archetype patterns are identified from practical experience or informal domain knowledge, and built *by generalizing recurring concepts in existing software analysis models (DP14)*. Therefore, Arlow and Neustadt also use the generalization as a pattern design principle.

Several characteristics of software patterns are summarized in (Winn & Calder, 2002) and we contemplate them as possible candidates for design principles. First, a characteristic that underpins all others is that a software pattern is generative. This means that it can have a number of concrete and different instances. Further, a software pattern (c1) implies an artifact, which means that patterns are building components for software models, (c2) bridges many levels of abstraction, from concrete to abstract artifacts, to provide for ease of problem understanding and problem solving, (c3) is both functional and nonfunctional[8], which means, respectively, that patterns express and balance possibility and feasibility, (c4) recognized and demonstrated in a concrete artifact, (c5) captures hot spots, which means that patterns capture invariant and variant parts of solutions and provide a structure to manage them, (c6) is a part of pattern language, which means that patterns should be combined if needed for designing the solution, (c7) is discovered and validated practically, rather than purely theoretically, (c8) is grounded in a domain to which it applies and may have no meaning and usefulness outside that domain, and (c9) resolves key issues in a particular area by capturing key concepts and important aspects of their interplay. From these characteristics, we derive the *generativity (DP15)* as a design principle saying that a pattern should be made in a way that it can be instantiated, or simply, that it should be a template. Also, we see again the *abstraction principle*, notably from the c2 characteristic *(DP16)*. The characteristics c1, and c3 through c9 are not seen as design principles by us. They just describe what the design pattern is, and do not reveal how it should be designed.

Although patterns for the software design phase such as those offered by (Gamma, Helm, Ralph, & Vlissides, 1995) are not our primary interest, it is worth mentioning principles for their design. A profound requirement in software design patterns is a reusable and flexible design of software implementations. In the object-oriented programming paradigm, the reusability and flexibility is achieved by applying the encapsulation abstraction, inheritance, delegation of responsibilities by object compositions, and polymorphism principles. Software design patterns for object-oriented software are results of applying those principles. The encapsulation *(DP17)* and again abstraction are principle applicable to software analysis patterns as well.

4.2.4 Representation of software analysis model patterns

Software analysis model patterns are also represented graphically and textually. For the object-oriented analysis model patterns, naturally, object-oriented notations are used. Coad (1992) used his own graphical notation to

[8] Nonfunctional here do not necessarily encompass technical or implementation aspect, but rather a general discussion of pattern's pros-cons that could help for the adoption or adaption of that pattern in different contexts.

represent object-oriented analysis model patterns. Fowler (1997) used Martin's & Odell's (1994) notation for object-oriented models but later switched to UML. Arlow and Neustadt (2004) used UML[9] to represent their patterns. The textual descriptions provide additional information to graphical representation of patterns. Interestingly, Arlow and Neustadt (2004) introduced a literate modeling for representing patterns. The literate modeling combines narrative text with UML models, for ease of understanding of patterns.

Computer-processable representations of the software analysis model patterns we investigated are not provided, to the best of our knowledge. However, Arlow and Neustadt proposed a component-based modeling approach based on the MDA (Model Driven Architecture) to use their archetypes and archetype patterns as parameterized and reconfigurable components for conceptual modeling. By following MDA principles, business archetypes and archetype patterns should be captured as models in UML, and serializable into XML Metadata Interchange (XMI) format to allow their use in different UML modeling tools. Using UML tools, the parameterized components can be further re-configured by adding new details, excluding optional elements, and combining components together into more comprehensive models. Then they may be instantiated in two ways - isomorphic or homomorphic – by defining MDA-based model-to-model transformations. An isomorphic instantiation creates one class for each archetype, while homomorphic instantiation creates more than one class for each archetype. Arlow and Neustadt define this feature as a *pleomorphism*, which is an adaptation of an archetype pattern into different forms that meet different specific requirements. More importantly, a pleomorph always adheres to the semantics of its base archetype pattern. A base archetype pattern describes the common semantics across all its pleomorphs.

4.2.5 Applicability to semantic modeling of the domain-specific information

As stated earlier, discussion of applicability of the software development patterns for semantic modeling of domain-specific information such as MSC information makes sense only for the software analysis model patterns. For the software analysis model patterns, their representation in an ontology language such as OWL DL is needed if we want to use them for semantic modeling of domain-specific information. However, even if provided in OWL DL, their further adaption could be needed to meet specific requirements of a specific domains, such as the requirements from Table 1 in case of the MSC domain. Only a few of the software analysis patterns could be adapted to capture some of the MSC information, assuming their OWL representation. For instance, Fowler's *Unit*, *Quantity*, *Measurement*, and *Observation* patterns in OWL representation could provide for 'What is the hole diameter size?' or 'What is the hole diameter size tolerance?' or 'What are typical production runs?' CQs. Possibly applicable are patterns from Arlow & Neustadt (2004) library for capturing measure units, quantities (measurements), and general concepts such as party, address, organization, product, or product catalog.

4.3 Ontology design patterns

Ontology design patterns (ODP, for short) are reusable "modeling solutions that encode best modeling practices" for recurring ontology design problems (Suarez-Figueroa M., Gomez-Perez, Motta, & Gangemi, 2012). "ODPs are ready-made modeling solutions for creating and maintaining ontologies; they help in creating rich and rigorous ontologies with less effort" (Manchester ODP Library, 2009). ODPs for OWL ontologies started to emerge practically as soon as OWL gained wider interest. Currently, there are few libraries of OWL ODPs: The W3C Semantic Web Best Practices and Deployment Working Group ODPs (W3C, 2005); The Manchester ODPs Library (2009) built on work in (Aranguren, Antezana, Kuiper, & Stevens, 2008); and the ODP Portal (http://ontologydesignpatterns.org/) built upon results from NeOn European FP7 project (Presutti, et al., 2008).

Prior to ODPs, the Semantic Patterns (Staab, Erdmann, & Maedche, 2001) for modeling on Semantic Web were introduced. The semantic patterns such as *Locally inverse relation*, *Part-whole* or *Local range restrictions* from (Staab, Erdmann, & Maedche, 2001) are commonalities of different semantic modeling languages for Semantic Web. In other words, the semantic patterns are a means to communicate machine-processable semantic structures at an epistemological level of representation to provide for their reusability across different Semantic Web ontology languages. Similar to ODPs is Knowledge Patterns work (Clark, Thompson, & Porter, 2000) and the Reusable Components (Clark & Porter, 1997), which are explained below.

4.3.1 Typology of ontology design patterns

According to the ODP classification provided in (Suarez-Figueroa M., Gomez-Perez, Motta, & Gangemi, 2012) and (Presutti, et al., 2008), ODPs can be grouped into six different types, each addressing different aspects in ontology development. The Structural and Content ODPs are among those types of most interest to the ontology design, hence, to our discussion.

[9] The UML is the de facto standard graphical modeling language for representing software patterns (www.uml.org)

Structural and Content ODPs

Structural ODPs include Logical and Architectural ODPs. According to (Suarez-Figueroa M. , Gomez-Perez, Motta, & Gangemi, 2012), Logical ODPs are compositions of logical constructs of an ontology language. They are useful in solving certain design problems when the ontology language does not directly support certain logical constructs. For instance, *N-Ary Relationship ODP* is a Logical ODP for capturing n-ary relations using OWL, which allows only binary properties. Architectural ODPs are compositions of Logical ODPs and define the overall shape of an ontology such as taxonomical organization[10]. According to the Manchester ODP library, which provides Logical ODPs, the ODPs can be grouped into (1) *Extension ODPs* that by-pass the limitations of OWL language (e.g. *N-ary Relationship ODP*); (2) *Good Practice* ODPs that obtain a more robust, cleaner and easier to maintain ontology (e.g., *Closure ODP*); and, (3) *Domain Modeling* ODPs as solutions for logical modeling problems in bioscience ontologies (e.g. *List structure ODP, Composite object chains ODP*). Logical ODPs are not explicity bound to any domain-specific vocabulary; they are abstract descriptions of ontology concepts, relations or axioms, and they have an empty signature[11].

Content, sometimes called Conceptual, ODPs are ontology components that capture foundational, core, or domain-specific concepts and their features. As such, they can provide ready-to-use and reusable ontology components for modeling and capturing ontology content. Content ODPs are explicitly bound to a vocabulary for a specific or universal domain. Their signature is non-empty. For example, *Place ODP*, *PeriodicInterval ODP* or *Person ODP* from the ODP Portal are typical examples of a Content ODP. Content ODPs may be instantiations of Logical ODPs or a composition of other Content ODPs. The idea of Content ODPs was introduced first in (Gangemi, 2005). Content ODPs are currently provided in the the ODP Portal only.

Other four types of ODPs

Besides Structural and Content ODPs, there are four other types of ODPs: Correspondence, Reasoning, Presentation and Lexico-syntactic. These types are of lesser importance to our discussion. Correspondence ODPs include Re-engineering and Alignment ODPs. The former provides solutions for transformation of relation database or XML schemas into ontologies; the latter provides solutions for creating semantic associations between ontologies. Reasoning ODPs, such as *Normalization ODP* in (Manchester ODP Library, 2009), can potentially enhance reasoning tasks. Presentation ODPs propose naming conventions for ontology elements to enhance usability and readability of the ontology from a human perspective. Lexico-syntactic ODPs, or Syntactic patterns in Blomqvist (2010), are patterns of linguistic structures that are automatically translatable into corresponding ontology elements. In particular, Lexico-syntactic and Re-engineering ODPs are not of interest to requirements in semantic modeling of MSC information. Reasoning ODPs are defined as common reasoning tasks such as classification and inference, not as reusable components for capturing ontology content. Presentation ODPs are important from a human perspective, but not essential either for achieving the consistency in ontology design or simplifying ontology development activities. Alignment ODPs are not design patterns for capturing ontology content, but rather mapping constructs for mapping between two or more ontologies.

Knowledge patterns vs. ODPs

Knowledge Patterns from (Clark, Thompson, & Porter, 2000) and Reusable Components from (Clark & Porter, 1997) are predecessors of Content and Structural ODPs. The Component Library (CLIB) introduced by Clark & Porter (1997) is a library of generalized Reusable Components that capture foundational concepts that can be used to construct knowledge-based systems. According to Clark & Porter, a Reusable Component "encapsulates a coherent system of concepts and their relationships." It can be composed with other reusable components and instantiated by binding its signature to objects in the target domain. A Knowledge Pattern (Clark, Thompson, & Porter, 2000) is a template that one can populate using a specific domain vocabulary and instantiate it into axioms representing a certain theory. In conclusion, Knowledge Pattern and Reusable Component notions are similar to the ODP notions, with the difference in formal representation. Reusable Components are implemented in a language called KM (Eilerts & Eilerts, 1994).

[10] Blomqvist (2010) in her ODP typology propose Architectural patterns too, in addition to Application, Design and Syntactic patterns. That typology is focused on the applicable levels of granularity of patterns with respect to the levels of structural abstraction of ontology.

[11] A design pattern signature is a graph representation of the design pattern with nodes and edges either named or not. A signature is a non-empty signature when all nodes and edges in the graph representation of design pattern are named; otherwise, a signature is an empty, when none of the nodes and edges are named, or a partly-empty signature when some of nodes or edges are left unnamed.

4.3.2 Vertical-horizontal variability of ODPs

Content ODPs also may vary both vertically and horizontally. For instance, the ODP Portal provides both universally applicable and domain-specific Content ODPs. In fact, the ODP Portal organizes Content ODPs according to the domain applicability. For example, *Transition ODP*, which represents basic knowledge about transitions, that include events, states, processes, and objects, is assigned to the manufacturing domain. In the portal, some of Content ODPs are vertical specializations and extensions of universally applicable or domain-specific Content ODPs. For instance, *Person ODP* is specialization of *Agent ODP*. Specialization means specializing some of the ODP elements, either classes or attributes. Any Content ODP can be further adapted and specialized to any specific context if needed. There are no restrictions in their use. In (Gangemi, 2005), the Content ODPs are classified into foundational and core ODPs, depending whether a Content ODP captures a foundational (universal) or a core (domain) concept. Structural ODPs, including Logical ODPs, in the analyzed libraries are all abstractions, devoid of any specific domain vocabulary; hence, universally applicable and with no vertical specialization.

4.3.3 Design principles for ODPs

According to (Blomqvist, 2009), there are at least two different approaches for constructing ODPs.

The first approach is to extract ODPs from existing ontologies or similar sources such as legacy libraries of reusable components. In fact, many of Content ODPs published in the ODP Portal are extracted from the DOLCE (Masolo, Borgo, Gangemi, Guarino, & Oltramari, 2004) foundational ontology. In (Gangemi & Chaudhri, 2009), the authors reported the creation of Content ODPs by translating CLIB components into OWL representations. Blomqvist (2010) reported on the creation of Content ODPs by translating the software analysis patterns from (Fowler, 1997) and data modeling patterns from (Hay, 1996) and (Silverston, 2001a) into OWL DL representations.

The second approach is to develop principles of good design and construct ODPs that reflect them. This approach is of interest to our research. Currently, such good design principles do not exist (Hammar, 2011b). Below, we provide ODP design criteria and characteristics that we extracted from the following papers: (Clark & Porter, 1997); (Clark, Thompson, & Porter, 2000); (Aranguren M. , 2005); (Gangemi, 2005); (D'Antonio, Missikoff, & Taglino, 2007); (Presutti, et al., 2008); (Blomqvist, 2010); (Hammar, 2011a), and (Hammar, 2014). We have considered those criteria and characteristics as possible principles for the ODP design and development. Let's highlight the main findings.

In, (Clark & Porter, 1997) and (Clark, Thompson, & Porter, 2000), the authors had characterized the Reusable Components and Knowledge Patterns as highly *abstract theories (the abstraction principle - DP18)*, which capture a *small number of concepts (DP19)*. In (D'Antonio, Missikoff, & Taglino, 2007), the OPAL ontology design patterns are also characterized as abstraction (*DP20*) and, thus, provided as parameterized *templates (DP21)*. Certainly, the abstraction, as we have discussed before for data modeling and software analysis patterns, may be adopted to be an ODP design principle. It means that an ODP captures a concept that is super-categorical for all subordinate concepts. The number of concepts, or the size of the model, is also selected before as the design principle, and may be adopted to be an ODP design principle too.

In (Aranguren M., 2005), ODPs, or more specifically, Manchester's Logical ODPs, are also characterized as abstractions that *hide OWL DL complexity (DP22)* and provide for efficient reasoning. Each Manchester's Logical ODP encompasses a few number of abstract concepts and structural-logical relationships; however, precise guide or quantification of the number of encompassed concepts is not given in (Aranguren M., 2005). Hiding OWL complexity may also be an ODP design principle, although a very generalized one, while the efficient reasoning is more a requirement.

Presutti et al., (2008) summarized characteristics of Content ODPs as follows (c1) requirements covering components, (c2) computational component, (c3) *small, autonomous* components for better diagrammatical visualization, (c4) *hierarchical* components based on specialization or generalization, (c5) *cognitively relevant* components that are intuitive and compact, catching relevant, "core" notions of a domain, (c6) *linguistically relevant* components that match linguistic patterns called frames from repositories such as FrameNet[12], (c7) *reasoning relevant* components that allow some form of inference, and, (c8) *best practices* components based on reusability in actual situations. Only some of these characteristics we considered as ODP design principles. In particular, the characteristic c3 "*small, autonomous" (DP23)* and C5 "*intuitive and compact...catching relevant, core notions*" *(DP24)* indicate a design principle regarding the size and conceptual coverage of the ODP. The c4 "*hierarchical*" characteristic is a general principle that emphasizes the vertical-horizontal variability of an ODP. For

[12] http://framenet.icsi.berkeley.edu/

us, the rest of them (the characteristics c1, c2, c6, c7 and c8) more tell about what the Content ODP is, than how to be designed or developed.

In addition to these characteristics, the authors also discuss *anti-pattern* characteristics that describe what a Content ODP should not be. For example, Content ODPs should not be a single isolated class or list of unrelated classes. They also should not be a single property with neither range nor domain defined, or list of such properties. The anti-pattern characteristics may be adopted to be ODP design principles *(*we refer to all them using id *DP25)*, as there are very precise in saying what should be avoided in an ODP implementation. OWL representation pitfalls summarized in (Rector, et al., 2004), such as absence of "closure axiom" to close off the possibility of further additions for a given property, are similar and can be contemplated as anti-pattern recommendations as well.

In Gangemi (2005), *intuitive and compact visualization* of Content ODP is pointed out as an essential ODP design principle. A Content ODP "requires a critical size, so that its diagrammatical visualization is aesthetically acceptable and easily memorizable" *(DP26)*.

In (Blomqvist, 2010), the author mentioned that granularity, sometimes called conceptual coverage should be such that the ODP *encompasses only concepts which are needed for one specific modeling problem, but with a small number of concepts so that all aspects can be visualized at the same time*. We adapt this as a design principle *(DP27)*. The author goes on to say that an ODP *follows OWL modeling best practices* such as specifying inverse properties, adding comments, applying naming conventions - also this statement can be considered as a ODP design principle *(DP28)*.

In Hammar (2011a), the author describes results of his work in developing a quality model for evaluating ODPs and emphasizes criteria including understandability, modifiability, minimal size, feasibility, completeness, expandability, performance, testability, ease of applicability, and documentation availability. The mentioned criteria are very general and as such they represent ODP requirements more than design principles to ODP development. From this work, we only take out a *minimal size while at the same time being feasible complete* ODP design principle *(DP29)*, as which we already derived in works in data model and software analysis model patterns.

Finally, Hammar (2014) recently studied effects of ODP design decisions on reasoning performances. From the literature review, Hammar in is work (2014) has identified several performance-affecting indicators, grouped into language expressivity profile indicators, inheritance hierarchy structural indicators, and indicators related to the logical axioms employed in an ontology. Upon the experimentations with effects that indicators may have on the reasoning performances, he provided several recommendations to the ODP design: (1) *avoid the use of multi-parent classes (DP30)*; (2) *limit the number of property domain and range definitions to the minimum required by the ODP requirements (DP31)*; (3) *use appropriate OWL 2 profiles to enhance performances (DP32)*; (4) *avoid designs that give a high count of ingoing/ongoing edges per node (DP33)*; (5) *limit the use of class restrictions. such as enumerations, property restrictions, intersections, unions, or complements to the minimum required by the ODP requirements (DP34)*; (6) *avoid developing ODPs that cause a deep subsumption hierarchy (DP35)*; (7) *limit the use of existential quantification axioms to the minimum required by the ODP requirements (DP36)*; (8) *rewrite general concept inclusion axioms into property restrictions if possible (DP37)*. All of them are very concrete recommendation for development of ODPs and therefore, we contemplate them as ODP design principles.

4.3.4 Representation of ODPs

ODPs are usually represented with graphical diagrams and textual descriptions. More specifically, Content and Structural ODPs in the ODP Portal, then the W3C ODPs and Manchester ODPs, are represented using UML or UML-like diagrams accompanied with textual descriptions. Those textual descriptions are usually structured into several parts such as the ODP name/title, functional specifications, implementation notes, consequences or side-effects, a source of the ODP, list of related ODPs, etc. The ODP Portal in most cases uses CQs to document the functional specification of their ODPs. The CQs are given informally using natural-language, which is probably not the best approach considering needs for ODP's search and retrieval.

Computer-processable representations of ODPs are not as rare as computer-processable representations of data model and software analysis model patterns. For instance, Content ODPs in the ODP Portal are provided as reusable building blocks as "small" OWL ontologies. The building blocks can be directly imported or copied into other OWL ontologies, and then adapted to meet needed requirements. To that end, there is a NeOn Toolkit (2012) for search, retrieval and specialization of the Content ODPs available from the ODP Portal. However, ODPs may be provided in other forms too, not necessarily as "small" OWL ontologies. For instance, D'Antonio, Missikoff, and Taglino (2007) have introduced *BusinessActor*, *BusinessObject*, *BusinessProcess*, *ComplexAttribute*, and *AtomicAttribute* ODPs as parametrized templates. D'Antonio et al.'s ODPs are foundational enablers of their OPAL (Object-Process-Actor Modeling Language) framework for modeling business ontologies.

The main difference between ODPs as parameterized templates vs. ODPs as building blocks is how they are used in an ontology development. The ODPs as parameterized templates must be first populated with values and then, if needed, instantiated or transformed into OWL. Whereas ODPs as building blocks are very small-scale OWL ontologies that can be imported or copied directly into an OWL ontology. The parameterized-template approach seems more flexible than the building-block approach. Moreover, for some types of ODPs, it is the only meaningful approach. While Content ODPs can take either the parameterized template or building block form, the Logical ODPs are not bound to any concrete domain terminology. Thus, logical ODPs are more appropriate in the parameterized template form. In fact, Co-Ode (Collaborative Open Ontology Development Environment) (2009) project provided all Manchester's Logical ODPs as parameterized templates as well as the parameterized template editor plugin for Protégé ontology management tool. That plugin uses OPPL, an Ontology Pre-Processing Language (Egana, Antezana, & Stevens, 2008), which is an abstract formalism for manipulating OWL ontologies at the level of graph patterns. A graph pattern is any recurring structure in an OWL graph, a notion different from the ODP notion.

4.3.5 Applicability to semantic modeling of domain-specific information

Good practices in using OWL or solutions to OWL limitations, such as Structural and Logical ODPs, may be applicable to semantic modeling of domain-specific information such as MSC information. However, the applicability will certainly depend on concrete domain-specific information requirements. As stated earlier, there is no available library of Content ODPs that could sufficiently cover the requirements of the reference MSC ontology. In particular, by reviewing specifications of the Content ODPs in the ODP Portal and CQs they claim to support, no satisfied coverage for functional requirements of reference MSC ontology was found. In the ODP Portal, only one Content ODP is categorized as specific to the manufacturing. That is a *Transition ODP*, which however, provides for capturing concepts not within the scope of MSC information. From that portal only those Content ODPs that provide for capturing structural organization of data such is a *Bag ODP* or *Collection ODP*, or for capturing a general concept such as *Party ODP, Location ODP are* potentially applicable in the specific field of our interest.

5 Summary of findings for domain-specific ODPs development

This section summarizes the analysis results of the paper. In particular, Sections 5.1 to 5.3 summarize desired design pattern characteristics, while 5.4 summarizes design pattern development principles that can be drawn from these characteristics. Section 5.5 generalizes the design pattern development principles, again from MSC domain-specific information perspective, but they apply also in other similar domains.

5.1 Typology of design patterns

We observed that the design patterns[13] can be categorized, in general, into the following three types.

First, **conceptual design patterns** capture universal or domain-specific concepts to provide components for building conceptual models of a concrete domain of interest. The conceptual design patterns can be means for capturing either information content or domain knowledge, or both, depending on the application needs. Typical examples of conceptual design patterns for capturing domain knowledge and information content are Content ODPs in (Gangemi, 2005) or Content ODPs in the ODP Portal. On the other side, logical and physical data model patterns such as Silverston's (2001a, 2001b) or Blaha's (2010) are examples of conceptual design patterns for capturing information content only. The conceptual design patterns can be often abstracted into structural design patterns, which are discussed next as design patterns that, in contrast to the conceptual ones, have empty or highly abstracted signature (which is a non-empty or partly empty signature with nodes and edges named using meta-level terms). A conceptual design pattern may be an instantiation either of a single structural design pattern or of the composition of several structural design patterns.

Second, **structural design patterns** capture structural concepts and structural relations between universal or domain-specific concepts such as hierarchical parent-child structures, taxonomies, aggregations, collections, ordered lists, bags, binary and n-ary relationship, and that capture logical structures which rely on logical relationships such as equivalence or disjointness. A typical example of a structural design pattern is *N-Ary Relationship* from (Hayes & Welty, 2006), which is also an RDF and OWL language-specific pattern. Another example would be a design pattern for capturing an ordered list (for example, in RDF or in ER language). The structural design pattern is not

[13] The data modeling, software analysis and ontology design patterns are hereafter collectively named as design patterns. They differ only in the information modeling paradigm in which they are used, but are essentiality the same notions.

bound to any specific- or general-domain vocabulary; it has an empty or highly abstracted signature, and always captures either structural concepts, structral relations or logical structures.

Third, **language-specific implementation recommendations** are syntactic patterns that are either 'best practice', ' syntactic sugar' or only 'available practice' in a concrete language for actual implementation of information or conceptual models. The design patterns as language-specific implementation recommendation are not modeling components that deal with domain concepts or their structural-logical relationships; they have no associated domain-semantics. Simply, those are encoding conventions that recommended how to encode models in a best way allowed by the constructs of the implementation language. Indeed, there can be a language-specific implementation recommendation for modeling certain structural-logical relationships, but we consider those recommendation as the structural design patterns (e.g. Hayes & Welty's *N-Ary relationship)*. A typical example of the design patterns as language-specific implementation recommendation is a *SynonymOrEquivalence* ODP from the ODP Portal. The *SynonymOrEquivalence ODP* suggests that two OWL classes (e.g., C1, C2) when identical and inside the same OWL ontology, should be merged into one OWL class having labels C1 and C2, rather than having two OWL classes and equivalence relationship among them. Hence, *SynonymOrEquivalence ODP* has no associated domain-semantic and is not capturing neither a structural nor logical relationship, therefore, is the OWL language-specific implementation recommendation only. Another examples are W3C's *Value Partition ODP* (Rector, 2005 and Manchester's *Good Practices ODP* (2009) such as a *Closure ODP*.

This three-type typology is a simplification of several typologies that other researchers introduced before. Our intuition is that such simplification might be very practical in organizing a library of MSC domain-specific ODPs.

5.2 Vertical-horizontal variability of design patterns

The vertical-horizontal variability is most noticeable for the conceptual design patterns, because general concepts may be specialized within and across different domains. On the other side, a structural design pattern typically has vertical variations only, since it represents general structural concepts not bound to specific domains. Implementation design patterns, viewed as encoding rules, have no vertical and horizontal variability. The main reason for applying the vertical-horizontal variability is to enhance reusability of design patterns within and across domains. The vertical-horizontal variability may also be useful in accomplishing centrally manageable design patterns. If, for example, the design patterns are organized into a taxonomy that has either horizontal or vertical specializations, these specializations will be allowed to also propagate down to the design pattern variations. Although it may be useful for organizing the design patterns, the variability may lead to complexity of relationships between design patterns. Our conclusion is that the vertical variability can be useful as a development principle to enable change management and reusability of domain-specific ODPs. However, the horizontal variability is not typical for domain-specific ODPs, as domain-specific ODPs may be very specific to the domain, which may impede cross-domain reusability.

5.3 Representation of design patterns

Design patterns can be represented using both graphical and textual description means, as discussed before in more details. Currently, UML is widely used for graphical representation of information models, and it is an appropriate notation to represent design patterns as well. MDA tools may provide serialized UML representations of design patterns to allow for their management and transformation into different languages. In particular, UML is well-suited for graphical illustration of the conceptual and structural design patterns, but not for the language-specific implementation recommendation. The design patterns as language-specific implementation recommendations are rather represented using textual descriptions.

Textual descriptions of design patterns typically state requirements that a design pattern satisfies, and provide a design pattern how-to guide. In a case of conceptual design patterns and functional requirements, competency questions (CQs) are typically provided, as in the ODP Portal. However, a limitation is that CQs are currently provided informally, using a natural-language form. In contrast to that, a formal representation of CQs, if available, would likely enhance the reuse of design patterns.

As we have seen from earlier discussion, design patterns are typically provided either as parameterized templates or as building blocks. In particular, conceptual design patterns can be provided in both of these ways, while structural design patterns are always parameterized templates. As a parameterized template, a design pattern is a class of possible contents with similar abstractions and of the same structure. To be applied, the template has to be instantiated and its parameters have to be populated with concrete values e.g., with domain-specific terms. On the other side, a design pattern as a building block is not parameterized; but rather, it can be directly imported or copied into a conceptual model, and then specialized, modified, and linked to other artifacts in that model.

In conclusion, domain-specific ODPs for semantic modeling of domain-specific information should be provided as parameterized templates. The reason is that they truly ensure modeling consistency rather than the building block approach which allows components to be arbitrarily altered.

5.4 Identified specific principles for design patterns' development

Our review of the works in design patterns identified no structured methodology for development of design patterns. Also, no specific metrics, neither qualitative nor quantitative, have been formalized yet to measure design patterns aspects such as competency questions satisfaction, reusability, and computational efficiency. Most of the currently available design patterns from the design pattern libraries mentioned in this paper, if not all, have emerged from practical experience. So far, the efforts have been focused on synthesizing collections of design patterns, not on synthesizing and structuring the design pattern development principles. Table 2 summarizes the design pattern development principles derived from the reviewed literature. Please note that we have merged some of derived principles (e.g., see 5th row of the Table 2), which we found to be essentially the same. Then, some principles in this table are, in fact, identified as characteristics of design patterns. However, they are paraphrased as principles; e.g., 'abstraction' characteristic is paraphrased into a principle saying 'abstract or generalize'.

Table 2 Derived specific principles in design pattern development

Derived Principles	**Is it applicable to development of domain-specific ODP?**
DP1.1 - encompass more than one, but less than ten concepts (Blaha, 2010)	Yes, but functional requirements have to be satisfied.
DP1.2, D1.3 - abstract enough to encompass a wide range of circumstances.. provide a starting point, rather than a complete solution (Hay, 1996)	Yes, but functional requirements have to be satisfied.
DP2 - adherence to Codd 3rd Normal Form (Silverston, 2001a)	No, as it is specific to ER models
DP3 - model with no-derive attributes (Silverston, 2001a)	Yes, if reformulated as 'no-inferable OWL attribute and relation' principle.
abstract or generalize [recurring] concepts and structures of universal or specific domains - revealed as the DP4 from (Silverston & Agnew, 2011), DP5 from (Blaha, 2010), DP11 from (Fowler, 1997), DP14 from (Arlow & Neustadt, 2004), DP16 from (Winn & Calder, 2002), DP18 from (Clark & Porter, 1997), and DP20 from (D'Antonio, Missikoff, & Taglino, 2007).	Yes
DP6. – Blaha's (2010) principles of avoidance DP6.1 - avoid symmetric relationship for data models (Blaha, 2010)	Yes, if reformulated into a 'avoid computational or implementation problematic relationships'
DP6.2 -avoid ambiguous naming conventions (Blaha, 2010)	Yes
DP6.3- avoid multiple inheritances (Blaha, 2010)	No, as it is specific to ER models
DP7 - adhere to a structure designed to meet the operational performability (e.g. STAR-schema structure for data analytics) (Silverston, 2001a)	Yes
DP8 - encompass only the concepts that reoccur in a particular domain or across domains (Coad, 1992)	Yes, but functional requirements have to be satisfied..
DP9 - make model as simple as possible, devoid of flexibility when flexibility unlikely needed (i.e. minimal model size, minimal coverage) (Fowler, 1997)	Yes, but functional requirements have to be satisfied. 'Free' adaptions are not allowed.
DP10 - encompass more than single concept, but no more than a few concepts ((Fowler, 1997)	Yes, but functional requirements have to be satisfied.
DP12 - capture universal, recurring concept in some domain (Arlow & Neustadt, 2004)	
DP13 - enable variability such that possible variations preserve core semantics (Arlow & Neustadt, 2004)	Yes
DP15 - make it generative [paraphrased from (Winn & Calder, 2002)]	Yes
DP17 - encapsulate actual implementation complexities (Gamma, Helm, Ralph, & Vlissides, 1995)	Yes
DP19 - encompass small number of concepts (Clark & Porter, 1997)	Yes, but functional requirements have to be satisfied.
DP21 - parameterize and template (D'Antonio, Missikoff, & Taglino, 2007)	Yes
DP22 - Hide OWL DL implementation/syntax complexities (Aranguren M., 2005)	Yes
DP23, DP24 - small, autonomous components for better diagrammatical visualization Intuitive and compact (Presutti, et al., 2008)	Yes, but functional requirements have to be satisfied.
DP25. – Presutti's principles of avoidance: DP25.1 - avoid design pattern as single disconnected, isolated single entities (Presutti, et al., 2008). This also appears in the DP6 (Blaha, 2010)	Yes
DP25.2 - avoid design pattern that encompass a simple class with single subclass (Presutti, et al., 2008)	Yes
DP25.3 - avoid design pattern that encompass a list of unrelated classes/properties (Presutti, et al., 2008)	Yes
DP26 - should have a 'critical' size, so that its diagrammatical visualization is aesthetically acceptable and easily memorizable (Gangemi, 2005)	Yes, but functional requirements have to be satisfied.

DP27 - encompass small number of concepts so that all aspects can be visualized at the same time (Blomqvist, 2010)	Yes, but functional requirements have to be satisfied.
DP28 - adhere to the (OWL) best modeling practices such as specifying inverse properties, adding comments, or applying naming conventions (Blomqvist, 2010)	Yes
DP29 - minimal size while at the same time being feasible complete (Hammar, 2011a)	Yes, but functional requirements have to be satisfied..
DP30 - avoid the use of multi-parent classes (Hammar, 2014)	Yes, but functional requirements have to be satisfied.
DP31 - limit the number of property domain and range definitions to the minimum required by the ODP requirements (Hammar, 2014)	Yes
DP32 - use appropriate OWL 2 profiles to enhance performances (Hammar, 2014)	Yes
DP33 - avoid designs that give a high count of ingoing/ongoing edges per node (Hammar, 2014)	Yes
DP34 - limit the use of class restrictions. such as enumerations, property restrictions, intersections, unions, or complements to the minimum required by the ODP requirements (Hammar, 2014)	Yes
DP35 - avoid developing ODPs that cause a deep subsumption hierarchy (Hammar, 2014)	Yes
DP36 - limit the use of existential quantification axioms to the minimum required by the ODP requirements (Hammar, 2014)	Yes
DP37 - rewrite general concept inclusion axioms into property restrictions if possible (Hammar, 2014)	Yes

We note that principles used to develop design patterns for one modeling paradigm may or may not be applicable to another modeling paradigm. For instance, Silverston's principle of no-derive attribute for data modeling patterns could be translated into a principle of no-inferable attribute/relation for ontology design patterns. Some principles, such as a minimal coverage (Fowler, 1997), may indeed be inapplicable for ontology design patterns, when consistency in ontology design is a requirement. The minimal coverage principle enables that design patterns serve as a starting point in modeling, which then may be freely adapted to specific additional requirements. Hence, having ontology design patterns developed according to the minimal coverage principle could easily lead into inconsistent design of ontology, unless 'free' adaptation is restricted. The third column in Table 2 indicates principles that we think are potentially applicable to development of domain-specific ODPs.

5.5 Generalization of principles for development of (MSC) domain-specific ODPs

To establish a library of domain-specific ODPs as reusable components for semantic modeling of domain-specific information, the identification of needed ODPs will be a first step, followed by the actual development of the ODPs by applying development principles.

Identification of needed ODPs, put in a context of MSC information modeling, means an identification and anticipation of reoccurring situations in MSC ontology design. In brief, the identification of needed MSC ODPs may be accomplished by identifying recurring information requirements within the given MSC information domain. The given MSC information requirements should be as comprehensive as possible to identify commonalities and recurring modeling situations. The recurring modeling situations may then be isolated as generalized reusable concepts and generalized reusable structures. If possible, that process should be semi-automated since the needs for new MSC ODPs are likely to emerge with the expansion of information requirements. Detailed discussion of the identification of needed domain-specific ODPs is out of scope of this paper.

Once the needed ODPs are identified, they should be developed according to certain principles. Possible principles for development of domain-specific MSC ODPs are summarized in Table 3 and discussed below. Table 3 is an adaptation of principles summarized in Table 2 to satisfy requirements identified in Table 1. In particular, it relates the MSC ODP requirements from Table 1 with a general principle (GP) introduced here. Where appropriate, we point to more specific Table 2 principles as possible approaches to the general principles.

Table 3 General principles to domain-specific ODP development: A MSC perspective

Id	**A requirement (from Table 1)**	**A general design principle to address the requirement**	**Possible specific principles (from Table 2)**
GP1	R.1. MSC ODP captures certain portions of MSC information to support MSC CQs	Scope of MSC ODP should be such that it covers interesting, recurring portion of MSC domain, as well as the semantics of that portion, including desired inferences.	DP1.1, DP1.2, D1.3, DP3, DP8, DP9, DP10, ,DP12,, DP19, DP23, DP24, DP25, DP26, DP27, DP29, DP30, DP31, DP33, DP34, DP35
	R.2. MSC ODP enables desired inference on MSC information		
	R.3. MSC ODP enforces MSC information validation		
GP2	R.4. MSC ODP provides for uniform design of MSC ontology	Establish naming and encoding conventions in OWL DL for capturing MSC information and its semantics.	DP6, DP25, DP28, DP30-DP37.
GP3	R.5. MSC ODP meets	Employ only certain OWL syntactic subset such as	DP3, DP6, DP7, DP28, DP30-DP37

	computational efficiency of MSC information communication	OWL 2 EL or OWL 2 RL to get the desired computational efficiency.	
GP4	R.6. MSC ODP enables computational efficient and straightforward MSC ontology mapping	Design an MSC ODP in a way that the need for additional alignment axioms during the ontology mapping process is reduced, and the increase of number of alignment axioms is not related to increase of instances in ontology.	There are no specific principles in Table 2.
GP5	R.7. MSC ODP can be re-used	Parameterize and abstract (or, generalize) an MSC ODP in order to provide an MSC ODP as a parameterized template.	DP1.2 , D1.3, DP4, DP5, DP11, DP13, DP14, DP15, DP16, DP17, DP18, DP20, DP21, DP22
GP6	R.8. MSC ODP is clear, understandable	Clearly document an MSC ODP i.e. to have names and purpose of encompassed elements unambiguous, to hide complexities, and to size the MSC ODP appropriately.	DP1.1, DP3, DP6, DP9, DP10, DP11, DP17, DP19, DP21, DP22, DP23, DP24, DP26, DP27, DP29, DP28, DP31, DP33, D34, DP35

- GP1. The scope or conceptual coverage of a single MSC ODP should include more than one, isolated concept or property. It should also include associated attributes and essential relationships of that concept. Otherwise, it is too trivial and it does not solve any modeling situation in semantic modeling of MSC information. Furthermore, it does not satisfy any of the established requirements. On the other side, a MSC ODP should not involve many concepts and properties, since that may impair its visualization, clarity, and operational aspects.
- GP2. The encoding conventions suggest the use of same OWL DL constructs, in a same way and order, for capturing essentially the same requirements. The encoding conventions may include some of existing OWL best practices reported in the ODP Portal at ontologydesignpatterns.org, in Manchester ODP Library (2009), in (W3C, 2005), or (Rector, et al., 2004). The encoding conventions should be such that any undesired inferences are prevented, since they may affect performances and correctness of MSC information retrieval. Defining the encoding conventions is a process of striking an optimal balance between the functional requirements, on one side, and nonfunctional requirements, on the other. The encoding conventions may employ only certain OWL syntactic subset to satisfy computational efficiency of MSC information communication, as discussed next.
- GP3. To address computational efficiency of OWL DL reasoners, OWL 2 specification has introduced three different OWL DL Profiles (W3C OWL WG, 2012). These profiles are OWL syntactic subsets that trade off different aspects of OWL's expressive power in return for computational benefits. For instance, OWL 2 EL profile enables polynomial time algorithms for all the standard reasoning tasks on very large ontologies. However, it places restrictions on use of OWL constructs that are not supported in OWL 2 EL. An MSC ODP should employ certain OWL syntactic subsets such as OWL 2 EL to get desired computational benefits. In addition to OWL itself, the computational efficiency may depend on the chosen DL reasoner. Existing DL reasoners differ in performances because of different implementation algorithms (Dentler, Cornet, ten Teije, & de Keizer, 2011). However, consideration of the efficiency of a particular DL reasoner might be beyond the ODP design, unless a requirement asks for a particular reasoner. Related to the computational efficiency are undesirable inferences, which should be clearly identified and disabled by the encoding conventions. For instance, it could be decided that certain OWL object properties should be defined without domain and range axioms. This happens in cases where a type of instances related with those properties will be explicitly asserted, so there is no undesirable type inference. It can also happen in cases where instances related with those properties may be inferred to potentially causing conflict with the instance declaration itself.
- GP4. One specific approach for this general principle was demonstrated in (Kulvatunyou, Lee, & Ivezic, 2013). In particular, two solutions for capturing range values such as Length Range or Width Range were demonstrated. First solution is a '`Range`' concept having '`hasValue`' data property for capturing a lower and upper bound of the range as one string value e.g. '0.5 - 1.5'. Alternative to that is a '`Range`' concept having two data properties as '`hasMinValue`' and '`hasMaxValue`' for capturing a lower bound and upper bound of the range separately, respectively. When the second approach was used in ontologies, then ontology-to-ontology mapping effort was significantly reduced and DL reasoning performances were increased, in comparison to the first solution. Hence, the second approach may be established as a specific design principle for modeling range values such as length, width or tolerance.
- GP5. The abstraction and parameterization are profound principles to the reusability. In practice, this principle is very simple and means that an MSC ODP should be developed as a parameterized template (i.e. parameterized component). To be applied, the parameterized component has to be instantiated and parameters

have to be populated with concrete values, e.g., MSC terms such as 'EDM', 'WireEDM', 'LaserMachine', or lower and upper bound of the range value, etc.

- GP6. End-users need to understand an MSC ODP, so it should be clearly documented with self-describing names. For example, a MSC ODP for capturing machining services such as EDM or CNC should be named specifically as a 'MachiningServiceODP'. Only if that ODP can be re-used for capturing non-machining services such as Prototyping, then it can be named more generically as 'ServiceODP'. Importantly, the understandability is tightly related to complexity of ODP, therefore, also to the trade-off between computational efficiency and the understandability. There can be modeling choices that are more understandable than others, but that instead impede computational efficiency, as reported in (Hammar, 2014). An ODP should not be encompassing too many concepts and properties, otherwise it may become hard to understand.

6 Conclusion and Future research directions

Semantic modeling of the domain-specific information in OWL DL faces as challenges consistent and quality development and evolution of a reference ontology of the domain. The domain-specific ODPs are envisioned as reusable components that may ease the reference ontology development and evolution while ensuring the consistency and uniformity of its design. However, as our review of the literature revealed, there is a lack of structured methodology and measureable criteria and principles for developing the domain-specific ODPs. In addition, there is a lack of methods for measuring quality of ODPs. To that end, this paper analyzed the literature, derived and summarized possible principles for development of the domain-specific ODPs and put them into an applied, MSC information perspective.

As a next step, concrete and specific design principles need to be created to refine and make operational the general principles. For example, a specific design principle may be to encompass more than one and less than ten concepts in a single ODP, to avoid inverse object properties and avoid universal restrictions, or to avoid domain and range axioms on properties to avoid complex types of inferences. Also, as future work, such design principles need to be validated. Finding and validating the specific design principles that address the domain-specific ODP requirements, such as the ones presented in this paper, is an especially challenging research task. We see that an experimentation using functional requirements as test cases and measurement of computational efficiency of an ODP is likely to be an only feasible way to identify and validate clear design principles for such requirements.

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

Collaborative Open Ontology Development Environment project. (2009, Jun). Retrieved March 25, 2013, from Co-Ode: www.co-ode.org

Manchester ODP Library. (2009, Jul). Retrieved March 25, 2013, from Manchester Ontology Design Patterns: http://www.gong.manchester.ac.uk/odp/html/index.html

NeOn Toolkit. (2012, April). Retrieved March 25, 2013, from NeOn Toolkit: http://neon-toolkit.org

Ameri, F., & Dutta, D. (2006). An upper ontology for manufacturing service description. *ASME*.

Aranguren, M. (2005). *Ontology design patterns for the formalisation of biological ontologies.* University of Manchester, Faculty of Engineering and Biological Sciences.

Aranguren, M. E., Antezana, E., Kuiper, M., & Stevens, R. (2008). Ontology Design Patterns for bio-ontologies: a case study on the Cell Cycle Ontology. *BMC Bioinformatics, 9*.

Arlow, J., & Neustadt, I. (2004). *Enterprise Patterns and MDA: Building better software with archetype patterns and UML.* Wokingham: Addison-Wesley.

Barker, R. (1990). *CASE method: entity relationship modelling.* Wokingham: Addison-Wesley.

Blaha, M. (2010). *Patterns of data modeling.* CRC Press.

Blomqvist, E. (2009). *Semi-automatic ontology construction based on patterns.* LN University.

Blomqvist, E. (2010). Ontology Patterns - Typology and Experiences from Design Pattern Development. *Linköping Electronic Conference Proceedings*, (pp. 55-64). Uppsala.

Chen, P.-S. (1976). The entity-relationship model - toward a unified view of data. *ACM Transactions on Database Systems, 1*(1), 9--36.

Clark, P., & Porter, B. (1997). Building concept representations from reusable components. *Proceedings Of The National Conference On Artificial Intelligence* (pp. 369--376). Wiley & Sons.

Clark, P., Thompson, J., & Porter, B. (2000). Knowledge patterns. *KR2000: Principles of Knowledge Representation and Reasoning*, 591--600.

Coad, P. (1992). Object-oriented patterns. *Communications of the ACM, 35*(9), 152--159.

Codd, E. (1972). Further normalization of the data base relational model. *In Data base systems*, 33-64.

D'Antonio, F., Missikoff, M., & Taglino, F. (2007). Formalizing the OPAL eBusiness ontology design patterns with OWL. *Third International Conference in Interoperability for Enterprise Applications*, (pp. 345--356). Madeira, Portugal.

De Nicola, A., Missikoff, M., & Navigli, R. (2009). A software engineering approach to ontology building. *Information Systems, 34*(2), 258--275.

Dentler, K., Cornet, R., ten Teije, A., & de Keizer, N. (2011). Comparison of Reasoners for large Ontologies in the OWL 2 EL Profile. *Semantic Web Journal, 2*(2), 71-87.

Egana, M., Antezana, E., & Stevens, R. (2008). Transforming the axiomisation of ontologies: The ontology pre-processor language. *Proceedigns of OWLED*.

Eilerts, E., & Eilerts, E. (1994). KnEd: An interface for a frame-based knowledge representation system.

Fowler, M. (1997). *Analysis Patterns: reusable object models.* Addison-Wesley.

Gamma, E., Helm, R., Ralph, J., & Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software.* Addison Wesley.

Gangemi, A. (2005). Ontology design patterns for semantic web content. *Proceedings of the Fourth International Semantic Web Conference* (pp. 262--276). Berlin: Springer.

Gangemi, A., & Chaudhri, V. (2009). Representing the Component Library into Ontology Design Patterns. *Workshop on Ontology Patterns International Semantic Web Conference.*

Gomez-Perez, A., Fernandez-Lopez, M., & Corcho, O. (2003). *Ontological engineering.* London: Springer.

Gruninger, M., & Fox, M. (1994). The role of competency questions in enterprise engineering. *Proceedings of the IFIP WG5, 7*, pp. 212--221.

Hammar, K. (2011a). DC proposal: towards an ODP quality model. *Workshop on Ontology Patterns, International Semantic Web Conference*, (pp. 277--284).

Hammar, K. (2011b). The State of Ontology Pattern Research. *Perspectives in Business Informatics Research: Associated Workshops and Doctoral Consortium*, (pp. 29--37).

Hammar, K. (2014) Reasoning Performance Indicators for Ontology Design Patterns. *Proceedings of the 4th Workshop on Ontology and Semantic Web Patterns -12th International Semantic Web Conference, CEUR Workshop Proceedings, 2014*

Hay, D. (1996). *Data model patterns: conventions of thought.* New York: Dorset House Publishing.

Hayes, P., & Welty, C. (2006). *Defining N-ary Relations on the Semantic Web*. Retrieved March 2013, from W3C: http://www.w3.org/TR/swbp-n-aryRelations/

Im, K., Lee, J., Kim, B., Peng, Y., & Cho, H. (2010). Conceptual framework of supplier discovery via ontology-driven semantic reasoning. *41st International conference on Computers and Industrial engineering.*

Jang, J., Jeong, B., Kulvatunyou, B., Chang, J., & Cho, H. (2008). Discovering and integrating distributed manufacturing services with semantic manufacturing capability profiles. *International Journal of Computer Integrated Manufacturing, 21*(6), 631--646.

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

Kulvatunyou, S., Lee, Y., & Ivezic, N. (2013). A framework for canonicalization of MSC information. *Submitted for publication*, 1-25.

Martin, J., & Odell, J. (1994). *Object-oriented methods.* Prentice Hall PTR.

Masolo, C., Borgo, S., Gangemi, A., Guarino, N., & Oltramari, A. (2004). *WonderWeb Deliverable D18: The WonderWeb Library of Foundational Ontologies.* WonderWeb Project.

McGuinness, D. L., & Van Harmelen, F. (2004, February). *OWL - Web Ontology Language* . Retrieved March 25, 2013, from W3C: http://www.w3.org/TR/owl-features/

Presutti, V., Gangemi, A., David, S., de Cea, G. A., Surez-Figueroa, M., Montiel-Ponsoda, E., et al. (2008). *D2.5.1: A Library of Ontology Design Patterns: reusable solutions for collaborative design of networked ontologies.* NEON project deliverable.

Rector, A. (2005, May). *Representing Specified Values in OWL: "value partitions" and "value sets"*. Retrieved March 2013, from W3C - Semantic Web Best Practices and Deployment Working Group: http://www.w3.org/TR/swbp-specified-values/

Rector, A., Drummond, N., Horridge, M., Rogers, J., Knublauch, H., Stevens, R., et al. (2004). OWL pizzas: Practical experience of teaching OWL-DL: Common errors \& common patterns. *Engineering Knowledge in the Age of the Semantic Web*, 63--81.

Silverston, L. (2001a). *The Data Model Resource Book, Volume 1, A Library of Universal Data Models for All Enterprises.* Addison Wiley.

Silverston, L. (2001b). *The data model resource book, Volume 2: A library of data models for specific industries.* Addison Wiley.

Silverston, L., & Agnew, P. (2011). *The Data Model Resource Book: Volume 3: Universal Patterns for Data Modeling.* Addison Wiley.

Staab, S., Erdmann, M., & Maedche, A. (2001). Engineering ontologies using semantic patterns. *Proceedings of the IJCAI-01 Workshop on E-Business & the Intelligent Web*, (pp. 174--185).

Suarez-Figueroa, M., Gomez-Perez, A., Motta, E., & Gangemi, A. (2012). *Introduction: Ontology Engineering in a Networked World.* London: Springer.

Taylor, D. (1995). *Business engineering with object technology.*

Uschold, M., & Gruninger, M. (1996). Ontologies: principles, methods and applications. *Knowledge engineering review, 11*(2), 93-136.

Verelst, J. (2004). *Variability in conceptual modeling.* University of Antwerp.

W3C. (2005, March). *Semantic Web Best Practices and Deployment Working Group*. Retrieved March 25, 2013, from W3C: http://www.w3.org/2001/sw/BestPractices/

W3C OWL WG. (2012, 12 11). *OWL 2 Web Ontology Language Document Overview*. Retrieved March 25, 2013, from W3C: http://www.w3.org/TR/owl2-overview/#ref-owl-2-profiles

Winn, T., & Calder, P. (2002). Is this a pattern? *Software, IEEE, 19*(1), 59--66