INFORMATICA, 2018, Vol. 29, No. 3, 439–466

DOI: http://dx.doi.org/10.15388/Informatica.2018.175

A Comparative Study of Approaches of Ontology Driven Software Development

Hele-Mai HAAV
Department of Software Science, Tallinn University of Technology
Akadeemia tee 15a, 12618 Tallinn, Estonia
e-mail: helemai@cs.ioc.ee

Received: October 2017; accepted: May 2018

Abstract. Ontology Driven Software Development (ODSD) combines traditional Model Based Software Development (MBSD) techniques with ontology technology in order to provide extensions to and advantages over MBSD. The goal of the paper is to identify current ODSD approaches and to provide qualitative and comparative analysis of the collection of identified approaches. Main research questions of the paper concern the ways of how ontologies are integrated to MBSD process and how their usage advances MBSD. Benefits and challenges of each of the discussed approaches are presented. The analysis is based on literature and projects reviews in the fields of ontology engineering, MBSD and ODSD. The result of the analysis provides understanding of what is the role of ontologies in ODSD and shows whether application of ontology technologies to the MBSD process gives rise to a new paradigm called consistency preserving software development or not.

Key words: ontology, model-based software development, MBSD, ontology driven software development, ODSD.

1. Introduction

In the Model Based Software Development (MBSD) process conceptual models of software are used for generation of code of a software system. Conceptual models of software are specified using formal or semiformal languages (e.g. Unified Modelling Language (UML)[1]). There is a wide range of MBSD approaches (OMG, 2003a; Gronback, 2009; Fritzson, 2014; Kalnins *et al.*, 2005, 2010) and tools that enable code generation from given software models (e.g. Modelica[2], MetaEdit+[3]). However, most software models used by these approaches do not have formal semantics and related logical reasoning facilities.

On the other hand, ontologies can be considered formal models of a domain of interest. They are expressed in languages like Ontology Web Language (OWL) (Motik *et al.*, 2012) that has underlying formal semantics expressible in Description Logics (DL) (Baader *et al.*, 2009) and supported by ontology reasoning services based on DL reasoning.

[1] http://www.omg.org/spec/UML/.
[2] http://www.modelica.org.
[3] http://www.metacase.com.

Combining traditional MBSD methods and ontology technology methods in the process of software modelling and development is called Ontology Driven Software Development (ODSD) (Assmann *et al.*, 2010; Pan *et al.*, 2013; Staab *et al.*, 2010; Gasevic *et al.*, 2009; Katasonov, 2012).

ODSD approaches exploit the expressive language for the representation of the knowledge of software modelling domain (e.g. OWL) and the powerful ontology reasoning (e.g. DL reasoning) (Parreiras and Staab, 2010). For example, standard ontology reasoners can be used for consistency checking, constraint validation, and query processing of software models. The knowledge that is described in ontology is separated from the execution logic of software system. This makes it possible to keep domain knowledge only in the ontology and query the ontology for getting specific knowledge of a domain (Gasevic *et al.*, 2009; Haav and Ojamaa, 2017; Hoehndorf *et al.*, 2009; Katasonov, 2012; Pan *et al.*, 2013).

Traditional MBSD approaches like the Model Driven Architecture (MDA) (OMG, 2003a) use domain models represented in UML for generation of code for specific software systems. In contrast, ODSD uses domain models in the form of ontologies not only for code generation, but also during run-time of the software system (Atkinson *et al.*, 2011).

ODSD is intended to provide advantages over traditional MBSD approaches and give rise to the consistency preserving software development where ontology reasoning and querying services are applied through the whole software development life cycle in order to preserve consistency of models (Pan *et al.*, 2013).

The main motivation of this paper is a lack of comprehensive comparative analysis of current approaches of ODSD and corresponding tools in order to understand the state of the art and future developments of ODSD. Existing surveys capture only some aspects related to ODSD as follows: comparison of meta-models and ontologies (Henderson-Sellers, 2011), roles of ontologies in software requirements engineering (Valaski *et al.*, 2016) and in domain specific languages (Sutii *et al.*, 2014) as well as ontology-based reuse of domain and enterprise engineering assets (Caplinskas *et al.*, 2003).

The goal of the paper is to identify fully developed approaches of using ontologies in the MBSD process and to provide comparative analysis of the collection of identified approaches in order to draw some conclusions with respect to the current level of the adoption of ODSD and consistency preserving software development.

Main research questions of the paper concern ways of how ontologies are integrated to MBSD and what their role in the MBSD modelling pyramid is. The paper also analyses the use of ontology reasoning. Benefits and limitations of each of the discussed approaches for MBSD are presented and compared. The analysis is based on the literature and projects' review in MBSD and ODSD fields.

The first contribution of this work is a review of the state of the art of ODSD approaches. The second contribution is an analytical comparison of ODSD approaches identified for this work. The third contribution is a discussion of the benefits and future challenges of ODSD with respect to consistency preserving software development.

The rest of the paper is structured as follows. Section 2 gives a brief overview of terminology of MBSD and explains modelling principles of MDA. Section 3 introduces ontology technology principles and gives an overview of efforts of integrating ontologies to

MBSD. Selection criteria and an analytical comparison of selected ODSD approaches are given in Section 4. Section 5 is devoted to related surveys on ODSD. Section 6 concludes the paper.

2. Model Based Software Development

Model Based Software Development (MBSD) is an improvement of the software refinement method of the 1970s where in addition to manual refinement (semi)automatic model transformations are used to derive from abstract models more concrete models of software artefacts. Reverse connections between these models are also important. The connections of model elements are achieved using meta-models that define sets of valid models. The basic advantage of MBSD is the improvement of software development process by using software models that abstract away certain implementation details and are much closer to the problem domain comparing to general purpose programming languages.

2.1. *Overview*

In order to promote the usage of MBSD approach, the Object Management Group (OMG) initialized standardization efforts and as a result in 2001 introduced Model Driven Architecture (MDA), an architectural framework for using models in software development (OMG, 2003a). The MDA approach is not new as using models as abstract representations of software artefacts in order to automatically produce the program code is not new. Also, the idea of building software models that are independent of any platform is not new. However, MDA is the first systematic approach that provides means for using models in different stages of software development.

Another effort to promote MBSD is Eclipse Modelling Framework (EMF) (Gronback, 2009). It is a modelling framework and code generation facility for building tools and other applications based on a structured data model. EMF provides tools and run-time support to produce a set of Java classes from a given model specification described in XML Metadata Interchange (XMI) format.

2.2. *Terminology*

In general, a model is a representation of a world that might be the real world or an abstract world (e.g. another model). It consists of statements in some language that follows a defined syntax (Abu-Hanna and Jansweijer, 1994). Models can view the world from different viewpoints. For example, the structural view describes the static semantics of a world and the behavioural view expresses its dynamic semantics.

In particular, the OMG MDA document (OMG, 2003a) defines the notion of a model of a system as follows: “A model of a system is a description or specification of that system and its environment for some certain purpose. A model is often presented as a combination of drawings and text. The text may be in a modelling language or in a natural language”.

MDA defines three types of models (also called stereotypical models) as follows: Computation Independent Model (CIM), Platform Independent Model (PIM) and Platform Specific Model (PSM) (OMG, 2003a).

According to the MDA standard, CIM just describes concepts related to a particular domain, but with no reference to the problem to be solved in that domain. PIM describes a particular system that solves a particular problem but in a technology independent manner, while a PSM describes how this system can be implemented using a given technology. Other similar classifications of models consider conceptual models, specification models and implementation models instead (Fowler and Scott, 1997).

In MDA, software development methods start from CIM and PIM and incrementally transform PIM to PSMs from what the final code is automatically generated. Refactoring from PSM to PIM is also necessary.

2.3. *Meta-Modelling Principle*

As already mentioned above, a model can represent other models. Models that represent a set of models are called meta-models. They describe (specify) valid elements (structure) of models (i.e. how a world should be modelled). A model is an instance of a meta-model. Meta-models can have a specific purpose or domain in which they are applied (Assmann *et al.*, 2006).

According to meta-modelling principle, meta-meta-models can be built, too. They represent meta-models and describe valid ingredients of meta-models. A meta-model is an instance of a meta-meta-model. Accordingly, a meta-meta-model represents a set of meta-models.

On the other hand, the notion of the model in MDSD is related to the notion of language of expression of a model (see definition above). For example, in the MDA standard all meta-models must be written in the Meta-Object Facility (MOF) language (OMG, 2002) to be MDA compliant.

In general, the meta-model is a model of a modelling language (with its abstract syntax and static semantics) to be used for describing concepts used for modelling the model (i.e. a domain model that is an instance of a meta-model). Consequently, the meta-meta-model defines a language for expressing meta-models. In MDSD, using explicitly given meta-models is highly important for automation of software development (e.g. code generation).

On the basis of meta-modelling principle, OMG introduced a meta-pyramid of models used in MDA and presented it in the ISO Information Resource Dictionary System (IRDS) standard (ISO, 1990). It contains four levels (Fig. 1) as follows: M0 level (objects/system instances), M1 level (models/instance specifications), M2 level (meta-model or language specifications), M3 level (meta-meta-model or modelling concepts level). The OMG meta-pyramid uses MOF as the meta-meta-model on the meta-meta-model level. In this mainstream modelling pyramid, the UML based meta-models are used.

The MDA models, CIM, PIM, and PSM are considered on M1 level (models). On M2 level they are specified by corresponding meta-models that are dialects of UML, enriching the UML core by profiles.

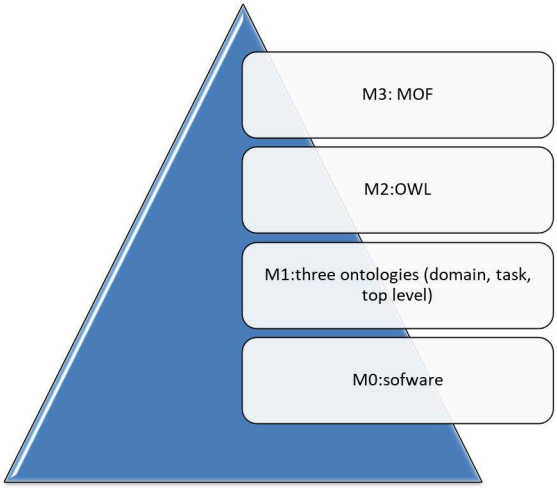

Fig. 1. The OMG meta-pyramid of models.

This meta-pyramid and MDA approach have been criticized in research community because of its complexity, inconsistency, volume, etc. (Assmann *et al.*, 2006). However, it is an industrial effort (i.e. a set of specific industrial technologies) and as such supported by various tools and used in software development practice.

In relation to Java, meta-modelling principles are used in the Eclipse Modelling Framework (EMF) project[4]. The core of the EMF framework includes a meta-model (the Ecore model) for describing models and run-time support for the models. EMF is based on two meta-models; the Ecore and the Genmodel model. The Ecore meta-model contains the information about the defined classes. The Genmodel contains additional information for the code generation. EMF allows to create the meta-model via different means, e.g. XMI, Java annotations, UML or an XML Schema. On the basis of the specified EMF meta-model the corresponding Java implementation classes can be generated. The generated code can be later manually extended.

3. Ontology Driven Software Development

A requirement to enhance software models to become formalized, complete, and precise introduced a new trend in MBSD that concerns integration of various ontologies as formal models to the MBSD process giving rise to ODSD. It refers to the different applications of ontologies in the software engineering process and in producing its artefacts (Pan *et al.*, 2013). The working groups at the OMG and the W3C have carried out initial steps towards ODSD in 2006 (OMG, 2006). Since then many researchers have provided different ways of the use of ontologies for improvement and enhancement of traditional soft-

[4]http://www.eclipse.org/modeling/emf/?project=emf.

ware engineering processes as well as of MBSD (Assmann *et al.*, 2010; Pan *et al.*, 2013; Staab *et al.*, 2010; Katasonov, 2012; Haav and Ojamaa, 2017).

3.1. *Ontology: Definition, Representation Languages and Reasoning*

Since the 2000s, the notion of ontology is widely used in order to represent domain knowledge to be commonly understood between humans, humans and computers or computers in various data and knowledge integration fields.

The term ontology has several definitions. In this paper, we use the definition given in computer science by Gruber as follows: "an ontology is an explicit specification of a conceptualization" (Gruber, 1993).

In general, an ontology includes computer-processable definitions of basic classes of things that exist in a domain and the relationships among them as well as the properties (or attributes) of these things.

Ontology needs to be specified formally for automated processing by computers. In the beginning of the 1990s, several formal languages have been developed for ontology representation. Currently, the most widely used ontology representation language is DL based OWL that is the W3C recommendation since 2004[5]. The latest version of OWL, OWL2[6] is a highly expressive language that allows for sound and complete calculi that are decidable as well as practically efficient (Guizzardi, 2013).

Ontologies represented in OWL enable different reasoning services that are based on underlying formal semantics of OWL, i.e. DL. The set of standard reasoning services provided by the most of DL reasoners (e.g. Fact++[7], Pellet[8]) is as follows (Baader *et al.*, 2009): consistency checking (checks contradiction among the definitions of concepts), satisfiability checking (finds all unsatisfiable concepts in a given ontology), subsumption computing (computes subclasses of a given class), classification service (classification of concepts according to subsumption of their definitions), and instance retrieval service (retrieves instances of a class).

Ontology querying service is provided using SPARQL[9] that is the W3C standard query language for Resource Description Framework (RDF)[10].

The above given reasoning services play an important role in the use of ontologies in ODSD as reasoning capability is one of the distinguished features of ontologies comparing to traditional software models used in MBSD.

3.2. *Comparison of Ontologies and Conceptual Models of Software*

Comparing to software models used in MBSD, ontologies can be considered a specific kind of models (Atkinson *et al.*, 2006; Henderson-Sellers, 2011; Guizzardi, 2007;

[5]https://www.w3.org/TR/2004/REC-owl-features-20040210.
[6]https://www.w3.org/TR/owl2-overview.
[7]http://owl.man.ac.uk/factplusplus.
[8]http://www.clarkparsia.com/pellet.
[9]https://www.w3.org/TR/sparql11-query.
[10]https://www.w3.org/RDF.

Assmann *et al.*, 2006). The works referred above identified the main differences between ontologies and models in MBSD as follows:

1. Models are oriented to realization but ontologies are not.
2. Ontologies are mostly intended to be used at run-time and they have formal representation that allows reasoning and querying. In contrast, models are basically used at design time and they do not provide reasoning services.
3. Models use the Closed World Assumption (CWA) while ontologies apply the Open World Assumption (OWA).

Analysing the semantics of OWL ontology and UML based models, we need to point out that OWL does not assume unique names for individuals (called Unique Name Assumption (UNA)). Therefore, in order to distinguish individuals by their name we need to explicitly state that they are distinct.

Another important difference of semantics is related to the set of instances; is it considered complete or not. The semantics of UML-based modelling assumes that the set of instances of a given model is complete (CWA). Therefore, the lack of information in instances of a UML-class based model is interpreted as negative information, since there is only one interpretation and everything that does not belong to this belongs to its complement (CWA). In contrast, OWL assumes incomplete knowledge by default and allows for validating incomplete models which are still in the design phase. The set of individuals, literals and property assertions has many different interpretations in OWL ontologies. Therefore, the absence of information in this set only indicates the lack of knowledge (OWA).

3.3. *Combining Ontologies with MBSD Towards ODSD*

The use of declarative knowledge representation in the field of MBSD is not new (e.g. recall the field of Knowledge Based Software Engineering in the 1980s).

Ontologies as semantic declarative models may extend a set of models used in the MDA. In the beginning of the 2000s, W3C issued the Ontology Driven Architecture (ODA) note that was a starting point of bringing together software engineering methodologies (e.g. MDA and UML) and semantic web technologies (e.g. RDF and OWL) (Tetlow *et al.*, 2006). This activity later resulted in issuing Ontology Definition Metamodel (ODM) by the OMG. The OMD is an OMG specification of application of concepts of the MDA to the ontology engineering in order to exploit features of UML tools for the creation of vocabularies and ontologies (OMG, 2003b). The latest version of the ODM is from the year 2014. Figure 2 depicts the hierarchy of the MOF-based meta-models for the OWL ontologies in the ODM.

M2 level defines OWL meta-model that is intended to be used for defining ontology models. Ontology of a particular domain belongs to M1 level and ontology instances to M0 level.

In general, there have been identified much more benefits of bridging semantic web technologies with the MDA than the ODM. A number of novel ways of extending capabilities of MBSD have been provided by many researchers (Assmann *et al.*, 2010;

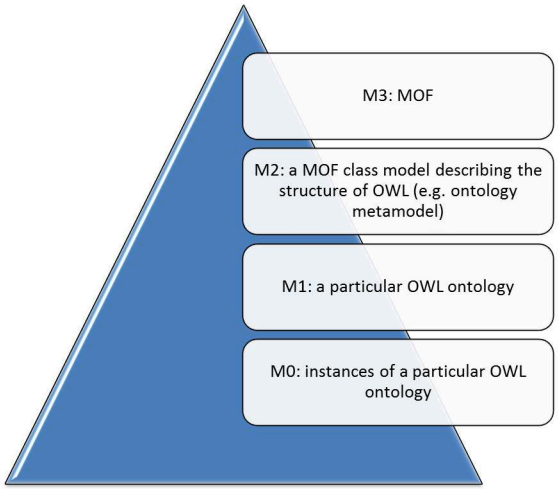

Fig. 2. The ODM hierarchy of models.

Pan *et al.*, 2013; Staab *et al.*, 2010; Katasonov, 2012; Haav and Ojamaa, 2017). These works have been basically motivated by the fact that MDA-based languages are not logic-based and do not enable reasoning. Therefore, their capabilities to describe the semantics of the domain are rather limited compared to ontology representation languages (e.g. OWL).

The most common way of applying ontologies in MBSD is to consider ontologies to be conceptual domain models, which are used in domain engineering to describe the problem domain that a software system should support (Guizzardi, 2013; Walter *et al.*, 2010; Staab *et al.*, 2010). For example, in Walter *et al.* (2010), ontologies are considered to be one single representation for meta-model and domain model. In this case, the terminological part (the TBox in DL) of the ontology consists of language concepts typically defined in the meta-model of a DSL. A set of instance assertions (the ABox in DL) relates individuals to the concepts described in the corresponding TBox.

In addition, in Falbo *et al.* (2002) an ontology-based approach to domain engineering is proposed. According to this method, a domain ontology is used as a domain model and domain analysis is replaced by ontology engineering. During the domain design stage the ontology is mapped to an object model.

Another broad area of application of ontologies in MBSD involves approaches that incorporate ontologies directly into software models themselves or use references to semantic metadata in software models or combine both approaches (Tetlow *et al.*, 2006).

4. An Analytical Comparison of ODSD Approaches

In this Section we provide the results of a comparative study that evaluates four identified complete ODSD approaches that exploit several ways of integration of ontology technology and MBSD. The goal is to discover the most distinguishing ways of the usage of

ontologies in ODSD enabling successful and applicable integration of both technologies. The analysis also aims at understanding whether state-of-the-art ODSD approaches may give rise to a new consistency preserving software development paradigm or not.

4.1. *The Research Methodology and the Selection of Approaches*

The analytical focus of this study is to identify similarities, differences and complementarity of the selected ODSD approaches according to the following aspects of interest:

1. Ways of integrating ontologies into the meta-modelling pyramid of MDA and their roles in ODSD.
2. The use of ontology services in ODSD.
3. Benefits of ODSD approaches beyond traditional MDA or MBSD.
4. Applications and limitations of ODSD approaches.

In order to be included into this analytical comparison task the ODSD approaches have been selected from a set of approaches we have found through extensive literature and project review applying the following selection criteria:

1. Availability of a method of systematic integration of ontologies into the meta-modelling pyramid of a particular ODSD approach.
2. Application of ontology reasoning services at run-time of a software system.
3. Availability of a tool that implements the ODSD approach and its application domain.

These selection criteria are related to the analytical focus of this study meaning that selected ODSD approaches may expose more characteristics than used for this study. After evaluation of the approaches we have identified four ODSD approaches that systematically utilize ontologies as software and domain models as well as provide some set of tools for ontology-based system modelling.

The most well-known approach among them is developed within the framework of the MOST project (Walter and Ebert, 2009; Pan *et al.*, 2013; Assmann *et al.*, 2010). It tries to bridge system modelling and ontological modelling in the field of software development. The main goal of the MOST approach is to enhance UML based syntactic modelling (structural modelling) by using OWL ontologies for representation of static semantics of software systems. This is done by providing transformations from MDA models to OWL and integration of these two technical spaces in software development process. Secondly, we analyse a hybrid ODSD framework (Katasonov, 2012) that is based on the main idea of the MOST approach but extends it with the usage of several types of ontologies and ontological services. It uses SPARQL patterns in addition to OWL (and DL) for ontological modelling. Third approach that is analysed in this paper is also a hybrid approach, where OWL ontologies are integrated into model-based software technology that uses automated program synthesis for generating software from models (Haav and Ojamaa, 2017). We call it the DSL meta-model ontology based approach in this paper. Last but not least, we analyse the three ontology method (Hoehndorf *et al.*, 2009) that uses domain, task and top-level ontologies for ontological modelling of a software system.

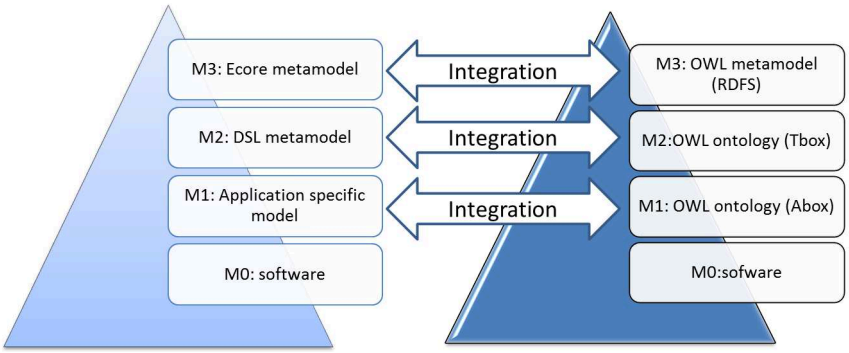

Fig. 3. The MOST model and language transformation approach.

4.2. *Integration of Ontologies into Meta-Modelling Pyramid*

In general, ODSD approaches exploit several kinds of ontologies representing knowledge of various domains. On the other hand, ontologies as descriptive models play different roles in the meta-modelling pyramid of MBSD and software architecture. Therefore, the integration of ontologies into the well-established meta-modelling pyramid of MDA is not a straightforward task to be solved by any ODSD approach.

4.2.1. *The Approach of the MOST Project*

The EU project MOST has been carried out by many researchers during several years and the results of the approach are well published in Staab *et al.* (2010), Walter *et al.* (2010), Pan *et al.* (2013), Walter *et al.* (2014).

The general goal of this approach is to improve software development by integrating ontology engineering into MBSD. The technology provided by the MOST approach is more general than the ODM described above (see Section 3.2). The MOST approach provides a platform independent solution to the integration of UML and OWL modelling.

Integration of MBSD and ontology engineering is considered on two levels in the MOST approach. First, software modelling language (i.e. UML) and ontology representation language (i.e. OWL) are integrated by providing a unified view of meta-models. These integrated meta-models can contain, in addition to software modelling constructs, also semantic descriptions and axioms based on OWL. Second, ontologies and models are integrated so that ontologies can be used in models, and vice versa.

According to the MOST approach, OWL meta-model described in RDFS[11] is located on M3 level of the ontology meta-modelling pyramid. Terminological part of OWL ontologies (TBox) is on M2 level corresponding to meta-models in MDA and instances of ontologies (ABox) belong to M1 level that corresponds to models in MDA metapyramid (see Fig. 3 at right hand).

The Most approach distinguishes between model transformations and model integrations. Model transformations automatically generate target models from source models

[11] https://www.w3.org/TR/rdf-schema.

using a set of transformation rules. First of all, the MOST approach provides transformations from the levels of Ecore meta-modelling pyramid to the levels of ontology meta-modelling pyramid as shown in Fig. 3 and described in more detail in Walter *et al.* (2014), Staab *et al.* (2010).

The Most approach defines 2 transformation bridges on M3 and M2 levels that could be used on M2 and M1 levels accordingly (see Fig. 3).

The transformation bridge on M3 level provides the transformation from software language constructs (e.g. Ecore) to the corresponding OWL constructs. Mappings between Ecore and OWL constructs are defined and later used by the OWLizer that is a program implementing transformations from Ecore meta-model or model into the OWL ontology TBox or ABox. One transformation takes the UML meta-model and the annotations as input and generates the corresponding OWL ontology TBox. Another transformation takes the UML model created by the UML user and generates individuals in the same OWL ontology (Staab *et al.*, 2010).

The transformation bridge on M2 level describes a transformation between models on M1 level.

In order to enable using models based on constructs of both modelling languages in a combined way (e.g. to integrate UML class diagrams and OWL) the MOST approach uses integration bridges (Staab *et al.*, 2010). These require existence of mappings between modelling concept on M3 level and between meta-models on M2 level belonging to both technological spaces. The MOST approach defines two integration bridges as follows (Staab *et al.*, 2010):

1. Metalanguage Integration Bridge is defined on M3 level and it provides an integrated meta-modelling language consisting of all classes of the Ecore meta-meta-model and OWL meta-model. It is used for designing language meta-models at M2 level with integrated constraints.
2. Language Integration Bridge is defined on M2 level between meta-models of the MDA meta-pyramid and OWL meta-modelling pyramid. This integration bridge is applied on M1 level for integrating software models and ontologies. A designer can use it for building integrated models that combine UML class diagrams and OWL.

4.2.2. *The Hybrid ODSD Framework*

The hybrid ODSD framework (Katasonov and Palviainen, 2010; Palviainen and Katasonov, 2011; Katasonov, 2012) extends the MOST approach by using four different types of ontologies and SPARQL patterns in addition to OWL for representing ontological knowledge.

The hybrid framework proposes to explicitly use the following types of ontologies for meta-modelling (Katasonov, 2012):

1. Domain ontology that describes the application domain of software.
2. Task ontology that describes platform independent problem-solving tasks that exist in the domain.
3. Software ontology that describes the software artefacts themselves, including the structural and the functional perspectives.

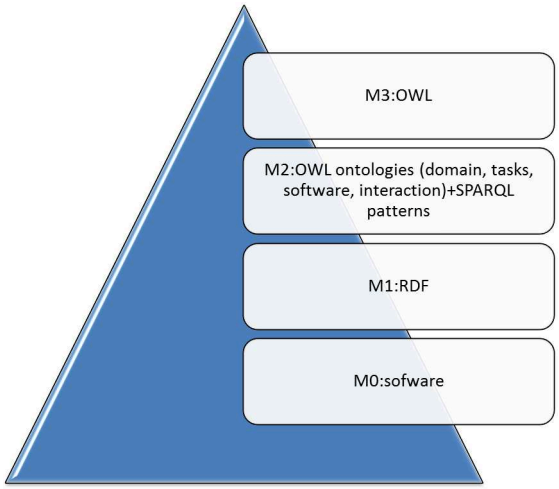

Fig. 4. The hybrid ODSD framework.

4. Interaction ontology that describes the interaction between software and its environment/domain.

The hybrid framework considers semantic descriptions of software on M1 level and concepts used for these descriptions are represented on M2 meta-model level as a set of ontologies (e.g. of four different types of ontologies listed above). Ontologies on M2 level can be defined using OWL or RDFS, which are located on M3 meta-meta-model level (see Fig. 4).

According to the semantic web technology, RDF is used to describe a set of individuals and relationships between them and therefore the hybrid framework assumes that M1 level models are given in RDF. Using RDF on M1 level makes it possible to exploit SPARQL patterns for encoding ontological knowledge as an alternative or extension to OWL. However, the hybrid framework does not replace OWL with SPARQL patterns as a whole, but only its mechanism for defining classes as restrictions (owl: Restriction class).

4.2.3. *The DSL Meta-Model Ontology Based Approach*

The DSL meta-model ontology based approach (Haav *et al.*, 2015; Haav and Ojamaa, 2017) uses a concept of a DSL meta-model ontology that links ontologies of different kinds for ODSD. This approach is an extension to the existing MBSD method (Kotkas *et al.*, 2011), where DSL meta-models are originally described in the CoCoViLa modelling language. The extension of the CoCoViLa incorporates OWL ontologies to M2 level of meta-models as depicted in Fig. 5.

The CoCoViLa modelling language on M3 level enables to describe meta-models of DSLs for various domains (see Fig. 5). A DSL meta-model ontology (its TBox and ABox) is created on M2 level of meta-modelling pyramid for each DSL to be developed and it integrates the following basic types of modular ontologies:

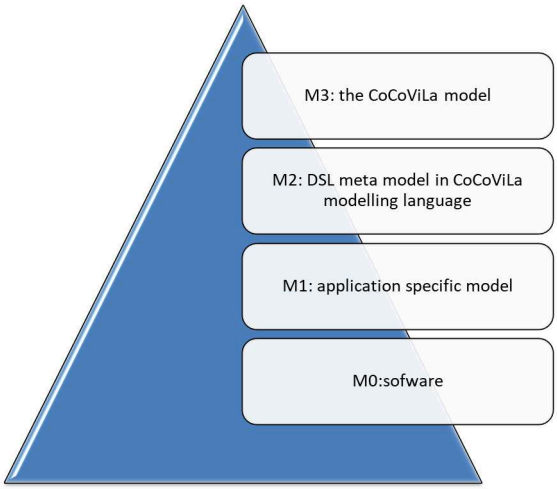

Fig. 5. The DSL meta-model ontology based approach.

1. The system ontology of a MSDB tool (e.g. CoCoViLa, Kotkas *et al.*, 2011) serves as a conceptual model of the MBSD tool (e.g. CoCoViLa) used for a DSL development and is a standard part of a DSL meta-model ontology that is imported to it. It describes concepts of a particular modelling language and the corresponding tool as well as relationships among them.
2. The domain ontology provides a specification of domain knowledge for a DSL. More than one domain ontologies can be linked if necessary. In principle, the DSL meta-model ontology can integrate additional ontologies for software modelling. For example, in Ojamaa *et al.* (2015) two additional ontologies were used for the attack tree DSL as follows: ontology of generic reusable components of the CoCoViLa simulation toolbox and ontology of a library of attack models.

Links to external software artefacts (e.g. several types of components of a DSL meta-model like Java classes from Java libraries, diagrams and their elements, the Java source code, etc.) can be represented in a DSL meta-model ontology using corresponding data property assertions or via the implementation relationship.

A DSL meta-model on M2 level is semi-automatically generated from the corresponding DSL meta-model ontology according to the predefined mappings from OWL to the CoCoViLa modelling language (Ojamaa *et al.*, 2015). This dynamic semantic composition of a DSL meta-model uses SPARQL in order to have access to the OWL descriptions of ontologies that are stored as RDF documents.

According to this approach, models on M1 level are automatically transformed to the corresponding valid logical representation in order to use the method of automatic construction of algorithm of a program (Mints and Tyugu, 1982) on M0 level and for efficient generation of the corresponding Java source code.

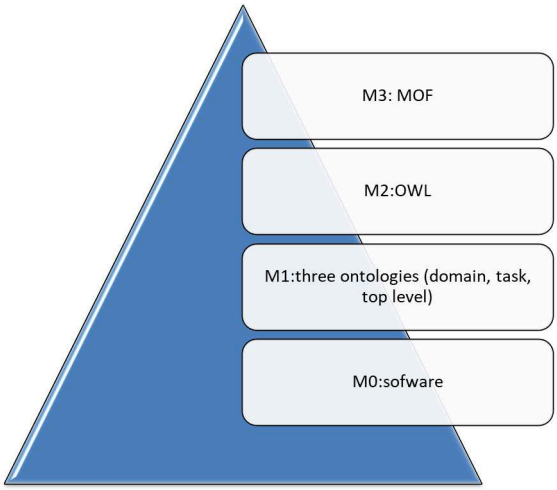

Fig. 6. Three ontology method.

4.2.4. *Three Ontology Method*

Three ontology method (Hoehndorf *et al.*, 2009) proposes to use three ontologies in order to define software models on M1 level as follows: a task ontology, a domain ontology and a top-level ontology. A task ontology (a conceptual model) captures the conceptualization of the problem domain that the software to be developed is intended to solve. A domain ontology represents domain-specific knowledge that is used by the software. A top-level ontology is a foundational ontology for both of these ontologies by providing foundation for concepts used in task and domain ontologies. Consequently, the task and domain ontologies are interrelated via top-level ontology and this makes it possible that a task ontology can be extended with the concepts and relations and axioms of a domain ontology.

According to the MDA modelling pyramid, three ontologies discussed above are located on M1 level (see Fig. 6). OWL is considered on M2 meta-model level. The authors of the three ontology method state in Hoehndorf *et al.* (2009) that MOF can be considered on M3 meta-meta-model level in case MOF specification of OWL is available.

The software implemented according to the three ontology method applies the task and the top-level ontologies to specify an interface for using entities from domain ontologies. Therefore, domain ontologies are replaceable modules in this software architecture and can be changed for the given task ontology. This makes it easy to adapt the software to a particular domain. Using ontologies as a part of software models enables the software to access its own ontological commitment (made explicit) during the run-time.

For example, the software can verify the semantics and integrity of data using these ontologies. The authors of the three ontology method argue in Hoehndorf *et al.* (2009) that it will be possible that the software model generated according to their method can be used for automated code generation.

Table 1
Ways of integration of ontologies to meta-modelling pyramid.

Level	The MOST project	Three ontology	Hybrid	Meta-model ontology
M3	OWL metamodel (RDFS)	MOF	OWL	The CoCoViLa model
M2	OWL ontology (Tbox)	OWL	Four OWL ontologies SPARQL patterns	DSL meta-model ontology and DSL specification
M1	OWL ontology (Abox)	Domain, task, top-level OWL ontologies	RDF	Application model
M0	Software	Software	Software	Software

4.2.5. *Conclusion*

On the basis of the previous analysis, we draw some conclusions about the similarities, differences, and complementarity of the ODSD approaches with respect to the integration of ontologies to the meta-modelling pyramid of MBSD.

All ODSD approaches under consideration use OWL for representation of ontologies and SPARQL for providing ontology query services. In addition, these approaches exploit ontologies as machine processable knowledge resources during the design and the run time of a software system.

There are significant differences among approaches related to kinds of ontologies that are exploited for software development as well as roles of these ontologies play in the meta-modelling pyramid of MBSD. The results of the analysis are shortly summarized in the Table 1.

The MOST approach basically transforms UML meta-models to corresponding OWL ontology for creation of ontologies that are translations of modelling languages themselves. Although it provides integration bridges for using combined models (e.g. UML combined with OWL), the scope of ontologies is restricted to with modelling language itself.

As seen from the Table 1, other three approaches provide wider range of ontologies that are used for software modelling on M2 and M1 levels as well as during the run-time of the software system. The hybrid framework uses four kinds of OWL ontologies on M2 level. The DSL meta-model ontology based approach exploits on M2 level rather flexible number of different ontologies that all are imported or linked to the DSL meta-model ontology and to its core component (i.e. previously defined system ontology of the corresponding modelling tool). The three ontology method uses three different ontologies on M1 level.

Concerning the ways of incorporation of ontologies into the meta-modelling pyramid of MBSD we may conclude on the basis of the Table 1 that the MOST approach follows ODE and has the similar hierarchy separating OWL ontology TBox and ABox to M2 and M1 levels accordingly. The same idea is picked up by the hybrid framework, which uses RDF data on M1 level that is basically the same as ABox. In contrast, the DSL meta-model ontology based approach and three ontology method do not separate OWL ontology TBox and ABox to be used at the different levels of modelling. However, these approaches differ in the levels these ontologies are exploited. The DSL meta-model ontology based approach uses ontologies on M2 meta-modelling level and the three ontology method on M1 modelling level.

4.3. *The Use of Ontology Services*

The use of ontology services is an important feature of any ODSD approach as this extends capability of ordinary MBSD approaches. Ontology services can be applied during design time of the software model as well as during the run-time of the software system that needs to use a DL reasoner for processing of the ontologies.

4.3.1. *The MOST Project*

The MOST approach runs the standard ontology reasoning services and SPARQL queries on an ontology (TBox and ABox) which is a representation of the meta-model of a modelling language and corresponding models created by the language user (Walter *et al.*, 2014; Staab *et al.*, 2010). This means that those services are integrated to M2 and M1 levels of model transformations. Accordingly, a model designer is provided with consistency checking service to examine does a model fulfil restrictions set by the ontology.

Satisfiability and subsumption checking services are applied on the checking the ontology itself, i.e. to check the consistency of the meta-model and to infer the hierarchy of defined classes that could be used to improve the corresponding meta-model.

Ontology querying using SPARQL provides the designer with an ability to query the model more flexibly than provided by standard modelling tools.

In addition to this, the MOST approach includes services that explain inferences in ontologies (could be used for debugging), services which notify a user about needed corrections to be made to ontologies in order to achieve consistent ontologies and services to link different ontologies (Walter *et al.*, 2014; Staab *et al.*, 2010).

4.3.2. *The Hybrid Framework*

In addition to traditional ontological services, the hybrid framework provides new ontological services as follows: semantic search in model repositories, semi-automated model composition services and policy enforcement service (Katasonov, 2012).

The hybrid framework allows semantic annotations of models of software components. Models are stored together with their semantic annotations in local or online repositories. This makes it possible to automatically discover models based on their annotations (i.e. perform semantic search) by model composition services (Katasonov, 2012). The hybrid framework provides the following model composition services (Katasonov, 2012):

1. Task-based model composition service enables to discover models of components on the basis of high-level tasks they realize and then incorporate these into the model.
2. Result-based model composition service implements a backward-chaining reasoning starting from the given resource that is to be produced as a part of the designed software. Next, using the semantic search service, models of components annotated as producing this kind of resource are retrieved. After that, the current model is searched for possible suppliers of the inputs required by these retrieved components.
3. Opportunistic-based model composition service provides a kind of forward chaining reasoning that analyses the current model with respect to what resources are being

available, and after that models of components that can consume those resources are searched for.

4. Policy enforcement service enables to check policies defined in relation to the designed software, i.e. on policies about allowed model compositions, not the modelling language as such.

4.3.3. *The DSL Meta-Model Ontology Based Approach*

The DSL meta-model ontology based approach uses standard DL reasoning services in order to ensure the consistency of a DSL meta-model ontology (Haav *et al.*, 2015; Haav and Ojamaa, 2017). The consistency of the DSL meta-model ontology is checked by using ontology inference provided by Apache Jena. This approach also uses SPARQL queries during the run-time of the software system. For example, queries are used to get an access to external software artefacts linked to the DSL meta-model ontology used by the software system and to obtain knowledge about the software model.

4.3.4. *Three Ontology Method*

The three ontology method uses standard ontology reasoning services on the software model that consists of three interrelated ontologies as follows: the top-level ontology, the conceptual model (software ontology) and the domain ontology (Hoehndorf *et al.*, 2009). The consistency checks of these ontologies are performed either for each of the component ontologies separately or for the combination of the top-level ontology and the conceptual model or the top-level ontology and the domain ontology. Inconsistencies are automatically detected and eliminated.

In addition to the model checking, ontology querying services are used during the run-time of the software system for querying the model of the software system itself. This enables the software to access the types, relations and constraints of its model.

4.3.5. *Conclusion*

All ODSD approaches under consideration use standard ontology services for consistency checking of ontologies related to software models on M2 and M1 levels. The approaches differ from each other on what ontology reasoning services are applied.

The MOST approach uses consistency checks on ontology of the meta-model of a modelling language and conforming models of the software system. Three other approaches apply consistency checks on a set of ontologies that are used for software modelling according to the respective approach.

Besides that, two approaches, the MOST approach and the hybrid framework, provide additional services. The MOST approach provides explanation, repairing and linking services. The hybrid framework enables services for semi-automated model composition and retrieval.

All the approaches provide ontology (model) querying service during the run-time of software system using SPARQL.

4.4. *Advantages of ODSD Approaches*

The OBSD approaches have a number of advantages over traditional MBSD. In the following analyses we take under consideration benefits that are pointed out by the authors of the ODSD approaches discussed in this paper.

4.4.1. *The MOST Project*

The authors of the approach of the MOST project consider to be the most important benefit of integration of UML and OWL giving to software developers possibilities to design and use more expressive models than previously (Pan *et al.*, 2013; Walter *et al.*, 2014). Specifically, when using the MOST approach software developers can combine object-oriented concepts and ontology concepts in a platform independent way.

They also underline that currently software development faces requirements of using and managing large and complex ontologies in many application fields like medical IS, multimedia and engineering applications, etc. (Staab *et al.*, 2010). To meet these new software requirements the integration of UML and OWL worlds is needed.

4.4.2. *The Hybrid Framework*

The authors of the hybrid framework (Katasonov and Palviainen, 2010; Palviainen and Katasonov, 2011; Katasonov, 2012) also emphasize that OWL ontologies as descriptive models provide greater expressiveness comparing to models created using meta-modelling tools like Ecore.

The hybrid framework basically utilizes SPARQL patterns instead of pure OWL. This has some technical advantages and extends expressive power of ontological modelling. Most important technical advantages include higher performance of using SPARQL patterns comparing to OWL as only RDF data storage supporting SPARQL querying is used. SPARQL queries are also useful for model checking purposes as they return all possible inconsistencies and do not stop upon discovery of the first one as DL reasoners do (Katasonov, 2012).

Concerning the expressiveness of ontological modelling, SPARQL patterns provide additional useful features that are not available in OWL. The authors of the hybrid framework (Katasonov and Palviainen, 2010; Katasonov, 2012) draw attention to at least four cases, where SPARQL patterns are very useful for ODSD. For example, OWL allows to define restrictions on the classes of RDF atoms (e.g. single model elements) while SPARQL patterns can be used for describing restrictions on non-atomic RDF structures (e.g. compositions of model elements).

4.4.3. *The DSL Meta-Model Ontology Based Approach*

The authors of the DSL meta-model ontology based approach point to the following general advantages of their approach as follows (Haav *et al.*, 2015; Haav and Ojamaa, 2017):

1. Applying ontology services on domain ontologies and on the DSL meta-model ontology is useful for debugging DSL meta-models.

2. Support for the distributed DSL development process is provided by using distributed artefacts (e.g. images, multi-media resources, linked data, etc.) linked to a DSL meta-model and used as components of a DSL.

In addition, the authors of the DSL meta-model ontology based approach evaluated specifically their work according to the following criteria: a level of alignment of domain knowledge captured in domain ontologies with DSL meta-models and independence of reusable knowledge from the internal representation of DSL meta-models (Ojamaa *et al.*, 2015). According to this work, the following benefits of ontology based meta-modelling approach can be distinguished:

1. A level of alignment of domain ontologies with DSL meta-models was improved in the structural parts of the specification of modelling concepts as well as building correct inheritance structures of concepts.
2. A level of independence of reusable knowledge from the internal representation of DSL meta-models was grown. The separation of different kinds of knowledge about the system, domain and a DSL into modular OWL ontologies makes the knowledge more reusable.

4.4.4. *Three Ontology Method*

The authors of the three ontology method underline benefits of their method over current MBSD approaches as follows (Hoehndorf *et al.*, 2009):

1. Providing a set of ontologically well-founded models and enabling ontology services on the conceptual model of the software and on the domain ontology.
2. The use of domain ontology as a module in a software model.
3. Application of ontology services during run-time of the software system. This is made possible due to the availability of the software model during the run-time of the software and can be used to verify constraints on data that is processed by the software.

4.4.5. *Conclusion*

It is generally recognized by the community of ODSD that OWL ontologies when used in MBSD make it possible to create more expressive software models than ordinary meta-modelling languages (e.g. Ecore). Application of ontology services on domain models as well as for verification of consistency of software models is considered to be another important advantage of ODSD approaches over MBSD.

Different ODSD approaches have their specific features that extend their modelling capability comparing to ordinary MBSD approaches.

The MOST approach provides a platform independent integration of UML and OWL (Staab *et al.*, 2010), the hybrid framework provides additional modelling power with using SPARQL patterns (Katasonov, 2012), the DSL meta-model ontology based approach supports the distributed ODSD by linking different artefacts (including ontologies) to DSL meta-model ontology (Haav *et al.*, 2015), and the three ontology method provides ontologically well-founded software model that is made available during the run-time of the software for application of ontology services (Hoehndorf *et al.*, 2009).

4.5. *Applications and Limitations*

Practical applicability and limitations are important characteristics of OBSD approaches providing information about what works and what does not work well when trying to apply ODSD to software development. Therefore, in this section we analyse tools implementing the corresponding ODSD approaches, their application domains and limitations of using a particular tool (and approach). According to our selection criteria, all the approaches under consideration are implemented within the framework of the corresponding tool.

4.5.1. *The MOST Project*

The MOST approach is implemented in the TwoUse Toolkit (Parreiras and Staab, 2010; Staab *et al.*, 2010), a model-based software development tool that enables developing software models with incorporated OWL ontologies and OWL ontologies that are related to software models.

The TwoUse Toolkit[12] has user profiles for model-driven software developers and for OWL ontology engineers. Therefore, the tool can also be used for ontology engineering. Model-driven software developers can use the toolkit for describing classes in UML class diagrams using OWL class descriptions, designing business rules using the UML Profile for Semantic Web Rule Language (SWRL[13]), extending software design patterns with OWL class descriptions and using some other OWL extensions to UML. In addition, ontology reasoning, explanation and query services are supported by the tool. The tool uses a SPARQL-like query language called SPARQLAS (Parreiras and Staab, 2010). The TwoUse Toolkit is implemented in the Eclipse Platform using the Eclipse Modelling Framework (Gronback, 2009).

The MOST approach is a general purpose ODSD approach and therefore it can be used for software engineering in many different domains (Parreiras and Staab, 2010). TwoUse and its principles have been used by its authors for different tasks within model-based software engineering field. For example, in Parreiras and Staab (2010) TwoUse principles are used for defining integrated meta-model of a DSL enriched by formal class descriptions in OWL in order to check the consistency of the model.

The TwoUse tool was also used for creation of a platform independent approach for ontology translation (Parreiras *et al.*, 2008) and for automatic generation of ontology APIs for semantic web applications (Parreiras *et al.*, 2009).

The authors of the TwoUse Toolkit point out the following limitations of their approach (Parreiras and Staab, 2010):

1. SPARQLAS queries may return OWL classes that are not part of the TwoUse model. This case needs to be handled separately.
2. From the point of view of TwoUse users, understanding of OWL and its semantics in addition to UML is important to be able to work with both modelling paradigms within one tool. Currently, model-based software developers are not experienced in working with OWL ontologies.

[12] http://code.google.com/p/twouse.

[13] https://www.w3.org/Submission/SWRL.

4.5.2. *The Hybrid Framework*

The hybrid framework is implemented as an extension of the model-driven software engineering tool called Smart Modeller that is a main component of the ontology-driven application development toolkit of SOFIA (Liuha *et al.*, 2009). A description of the modelling language used in Smart Modeller and some technical details of this tool can be found in Katasonov and Palviainen (2010), Palviainen and Katasonov (2011).

Smart Modeller defines a domain-specific modelling language that meets needs of SOFIA and provides a graphical editor for that language implemented in Java using Eclipse Graphical Modelling Framework (GMF).[14] The editor of Smart Modeller enables the developer to create a model of an application (presented as a directed graph consisting of elements and connectors) and then to automatically generate executable programming code for it. In Smart Modeller, the reuse of software components and models is supported by repositories, which are RDF data stores.

The implementation of the hybrid approach includes extensions of Smart Modeller providing the classification and model consistency checking services, the repository mechanism, three model composition services, as well as the policy enforcement service. Processing of RDF, SPARQL querying, and RDF-S reasoning are handled in Smart Modeller by exploiting OpenRDF Sesame[15].

Smart Modeller and its extensions are used for building applications for smart environments (Katasonov and Palviainen, 2010) and for implementation of home automation system and for a personal assistant application (Palviainen *et al.*, 2014).

One limitation that is mentioned by the authors of the approach concerns using SPARQL patterns instead of OWL for modelling (Katasonov, 2012). Inheritance over subsumption hierarchy cannot be expressed when using SPARQL patterns because these should be complete and defined in terms of the ground data on which the query is intended to be run. The hybrid approach overcomes this limitation by using a combination of SPARQL patterns and DL based approaches.

OWL constructs can be used but a patterns pre-processing engine is exploited for constructing the full pattern for a class before it is used in a query. This means that OWL construct for defining classes as restrictions (owl: Restriction class) is replaced by SPARQL pattern.

4.5.3. *The DSL Meta-Model Ontology Based Approach*

The DSL meta-model ontology based approach is prototypically implemented as an extension to the current version of the model-based software development tool CoCoViLa[16] Haav *et al.* (2015). Modelling and implementing DSLs with CoCoViLa can be done by diagrammatically defining different elements of a DSL and their interactive aspects as well as by using the textual specification language and Java. The CoCoViLa extensions provide developers with domain ontology-driven DSL modelling facilities.

[14] http://www.eclipse.org/modeling/gmp/.

[15] http://www.openrdf.org/.

[16] http://cocovila.github.io/.

The CoCoViLa extension is complemented with the CoCoViLa system ontology as a resource that formally describes the CoCoViLa modelling language and system concepts. This ontology can be reused as a part of a DSL meta-model ontology for the development of different kinds of DSLs.

The CoCoViLa extension helps to improve the DSL development process. A DSL application for solving a particular problem is semi-automatically done by means of the CoCoViLa tool. For that the DSL meta-model ontology is pre-processed in order to convert it to the internal structure of a DSL meta-model using Apache Jena tools and SPARQL queries. The CoCoViLa extension has been used in the domain of the IT security risk analysis for building threat modelling tools for educational purposes in IT security study programs (Ojamaa *et al.*, 2015).

One of the limitations of the approach is related to its implementation that is tightly related to the existing CoCoViLa system and its modelling language. This in turn is connected to another limitation that concerns different expressive power of DL that is a basis of OWL and a subset of Intuitionistic Propositional Calculus (IPC) that is a basis of the semantics of the CoCoViLa modelling language and program synthesis method used by the tool (Mints and Tyugu, 1982). This leads to the limited ability of the approach to capture full knowledge from OWL domain ontologies as IPC is less expressive than DL.

The semantic integration of artefacts from external tools and models into the CoCoViLa extension requires the commitment to a common system ontology or availability of system ontologies of external tools. In addition, the DSL meta-model ontology should be developed for building a DSL meta-model within the framework of a particular tool. The authors of the approach also mention that there is a lack of ontology engineering skills among model-based software developers limiting the exployment of ODSD methods (Ojamaa *et al.*, 2015).

4.5.4. *Three Ontology Method*

The three ontology method has been applied to the development of the ontology based semantic wiki in the domain of biology (Hoehndorf *et al.*, 2006). The wiki is called BOWiki and it is used for the annotation of genes and gene products with terms from ontologies. The conceptual model for the BOWiki (the task ontology) is a part of the top-level ontology GFO (Herre *et al.*, 2006). As a domain ontology (i.e. biological ontology) the biological core ontology GFO-Bio (Hoehndorf *et al.*, 2008) is used.

The authors of the three ontology method emphasize the following limitation of their approach (Hoehndorf *et al.*, 2009). This concerns the performance of DL reasoners. The software that is developed according to their approach needs to invoke a DL reasoner many times during the run-time providing ontology services to its operations. This slows down the overall performance of the software system. Although the performance of DL reasoners is improving, the authors of the method see that this is currently a bottleneck of using their approach for developing software that requires high performance.

4.5.5. *Conclusion*

The ODSD approaches analysed in this paper are implemented in original tools or as extensions to the existing MBSD tools.

Table 2
Tools, their application domains and specific limitations.

	The TwoUse extension	The Smart Modeller extension	The CoCoViLa extension	The BOWiki software
Application domains	Definition of integrated DSL meta-models, generation of ontology APIs	Smart environments and applications	IT security risk analysis	Semantic wiki
Specific limitations	Special handling of SPARQLAS queries (see Section 4.5.1)	Limitations related to SPARQL patterns	Inability of transforming all ontological knowledge to object system	Low runtime performance due slow DL reasoning

The MOST approach is implemented as the TwoUse toolkit intended for software development and ontology engineering providing UML means extended with OWL. The three ontology method is implemented to be used for development of a particular type of applications (i.e. semantic wikis).

The hybrid approach and the DSL meta-model ontology approach are implemented as extensions to the existing MBSD tools: Smart Modeller and CoCoViLa, accordingly.

It seems that the performance of DL reasoners do not satisfy the requirements of ODSD, especially when used during the run-time of the software system. Three approaches from four exploited specific methods to overcome this bottleneck. The Smart Modeller extension uses SPARQL patterns for modelling (Katasonov, 2012). The CoCoViLa extension only loads and validates the DSL meta-model ontology before transforming it to internal structure and uses SPARQL queries during the run-time of the software system (Haav and Ojamaa, 2017). The TwoUse uses SPARQLAS queries (Parreiras and Staab, 2010). Except for the hybrid approach, all other approaches use RDF documents (files) for storing ontologies but not RDF stores. This may explain slow reasoning services. Modern RDF stores provide fast and scalable reasoning services.

The authors of the TwoUse and the CoCoViLa extension mentions also that developers need knowledge of OWL in addition to traditional MBSD languages (e.g. UML) and usually they do not have it (Haav *et al.*, 2015; Parreiras and Staab, 2010).

Tools, their application domains and specific limitations are summarized in the Table 2.

5. Related Work

Most of the related work has already been discussed and cited in Sections 2 and 3. In this Section we consider and refer to related surveys. The application of various kinds of ontologies in software engineering process life cycle is analysed in Happel and Seedorf (2006). This work shows that ontologies could be or are used in almost all stages of software engineering. They provide a categorization of the usage of ontologies in the software engineering process including the run-time and the development time of a software system. They also looked at the kind of knowledge the ontology actually describes. As a

result, they distinguished between the problem domain that the software system tries to tackle and infrastructure aspects to make the software or its development more convenient. Their main conclusion was that the most important benefit of using ontologies in software engineering is related to reusing of domain knowledge through the software engineering life cycle.

Ontologies play important roles in software requirements engineering. According to the results of a systematic review in Valaski *et al.* (2016), ontologies have an effective role in the analysis, specification and elicitation activities. It is considered that ontologies have potential to be also applied in the negotiation and validation activities. Ontologies are more widely applied to conceptual understanding of the domain related to producible software. In addition to the requirements engineering, there are many ontology proposals applied to various kinds of models that are not only related to the requirements analysis, but are important to the model transformation in the software design phase. According to this survey, the most important contributions of ontologies are as follows: identifying problems in specification and models, improving communication, building more complete models, allowing traceability among artefacts and improving the quality of requirements identification.

A survey of two approaches of combining standard domain engineering techniques with ontology technology in order to reuse domain and enterprise engineering knowledge is provided in Caplinskas *et al.* (2003). According to the results of this work, both analysed approaches are not fully developed to be used for solving practical problems.

Although Valaski *et al.* (2016) identified a great number of ontology proposals, there was no top-level ontology that integrated knowledge of all the software engineering artefacts. Usually, ontologies are developed for a specific software engineering field and they are not sufficiently interrelated (Calero *et al.*, 2006; Souza *et al.*, 2013). In order to overcome this limitation SEON (a Software Engineering Ontology Network) initiative (Ruy *et al.*, 2016) tries to achieve consistent software engineering ontologies including core ontologies for software and software processes as well as domain ontologies for the main technical software engineering subdomains, namely requirements, design, coding and testing.

There is a literature review of using ontologies in DSLs (Sutii *et al.*, 2014) that is tightly related to this paper. The goal of their paper is to analyse what value application of ontologies creates to DSL development. The main conclusion of their paper is that ontology technology complements DSL development with formal description of concepts and relationships among concepts as well as with the reasoning services. However, the authors point to the complexity of integration of ontologies and DSL meta-models and models because this involves manual work that cannot be easily automated.

Our comparative study in this paper differs from surveys discussed above in that we consider fully developed ODSD approaches that also are implemented and applied to software development in some problem domain. This enables us to discover most important advantages and limitations of ODSD in order to understand whether ODSD may grow to become a new consistency preserving software development approach.

6. Conclusions and Discussion

In this paper we provided a comparative analysis of fully developed ODSD approaches that met our selection criteria. The approaches were searched out from projects and research papers. We found four ODSD approaches that met our requirements. These approaches have been analysed according to their methods of integrating ontologies into meta-modelling pyramid of MDA, application of ontology services, benefits and limitations comparing to traditional MBSD.

Findings of analysis of each of the features have been summarized in the corresponding sections. Therefore, in this section we draw only some larger conclusions and discuss some issues.

There are significant differences between ontology technology and ordinary MBSD technology (e.g. standard MOF based techniques). However, when combined in a smart way these technologies can provide ODSD frameworks, where both technologies complement each other and give some advantages in MBSD.

Most important advantage of ODSD over MBSD is the ability to create more expressive software models than ordinary meta-modelling languages (e.g. the MOST approach). Another advantage is the provision of access to ontologies as rich declarative models during the runtime of the software system in order to handle user interfaces or to control the behaviour of an application (e.g. Smart Modeller, BOWiki or CoCoViLa applications). In contrast, ordinary MBSD approaches provide access to their models only at design time of the software system. In addition, ontology technology provides better support for model checking using logical inference, integration of various artefacts of software development and interoperability of different models and systems than ordinary MBSD using MOF-based languages.

On the other hand, requirements of new semantic applications related to run-time access of models draw attention to unsatisfied performance of current DL reasoners. Three approaches from four exploited specific methods to overcome this bottleneck. In addition, there is a lack of ontology technology skills among model-based software engineers in order to work with both paradigms (i.e. ontology engineering and software engineering) within one ODSD framework.

On the basis of applications of ODSD approaches analysed in this paper, we may say that ODSD approaches are not yet widely accepted by industrial software engineering community as applications reported in the corresponding papers are not industrial ones. However, ODSD has impact to MBSD as it raises the level of abstraction of software models and expands the use of formal methods (e.g. reasoning).

Findings of this study do not convince us that a new consistency based software development paradigm will take place in near future because full integration of ontology technology and MBSD still needs more research, methodological and technological issues to be solved. Currently, software developers do not accept ODSD as a mature technology.

Acknowledgements. This work was partially supported by the Institutional Research Grant IUT33-13 of the Estonian Research Council.

References

Abu-Hanna, A., Jansweijer, W. (1994). Modeling domain knowledge using explicit conceptualization, *IEEE Expert*, 9(5), 53–64.

Assmann, U., Zschaler, S., Wagner, G. (2006). Ontologies, meta-models, and the model driven paradigm. In: Calero, C., Ruiz, F., Piattini, M. (Eds.), *Ontologies for Software Engineering and Software Technology*. Springer, pp. 249–273.

Assmann, U., Bartho, A., Wende, C. (Eds.) (2010). In: *Reasoning Web. Semantic Technologies for Software Engineering. Tutorial Lectures*. *LNCS*, Vol. 6325. Springer.

Atkinson, C., Gutheil, M., Kiko, K. (2006). On the relationship of ontologies and models. In: *Proceedings of the 2nd International Workshop on Meta-Modelling (WoMM 2006). LNI*, Vol. 96. GI, pp. 47–60.

Atkinson, C., Kennel, B., Goss, B. (2011). Supporting constructive and exploratory modes of modeling in multi-level ontologies. In: *Proceedings of ISWC2011*. Online http://iswc2011.semanticweb.org/fileadmin/iswc/Papers/Workshops/SWESE/1.pdf.

Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider P.F. (Eds). (2009). *The Description Logics Handbook*. Cambridge University Press.

Calero, C., Ruiz, F., Piattini, M. (2006). *Ontologies for Software Engineering and Software Technology*. Springer Science and Business Media, Heidelberg.

Caplinskas, A., Lupeikiene, A., Vasilecas, O. (2003). The role of ontologies in reusing domain and enterprise engineering assets. *Informatica*, 14(4), 455–470.

Falbo, R., Guizzardi, G., Duarte, K.C. (2002). An ontological approach to domain engineering. In: *Proceedings of the XIV International Conference on Software Engineering and Knowledge Engineering (SEKE-2002)*. ACM Press.

Fowler, M., Scott, K. (1997). *UML Distilled – Applying the Standard Object Modeling Language*. Addison-Wesley-Longman.

Fritzson, P. (2014). *Principles of Object Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach*. John Wiley and Sons, Inc.

Gasevic, D., Djuric, D., Devedzic, V. (2009). *Model Driven Engineering and Ontology Development*. Springer, Berlin.

Gronback, R.C. (2009). *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit*. Addison-Wesley Professional.

Gruber, T.R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2), 199–220.

Guizzardi, G. (2007). On ontology, ontologies, conceptualizations, modeling languages, and (meta) models. In: *Databases and Information Systems IV: Selected Papers from the Seventh International Baltic Conference on Databases and Information Systems (DBIS 2006)*, *FAIA*, Vol. 155. IOS Press, pp. 18–39.

Guizzardi, G. (2013). Ontology-based evaluation and design of visual conceptual modeling languages. In: Reinhartz-Berger, I., *et al.* (Eds.), *Domain Engineering, Product Lines, Languages, and Conceptual Models*. Springer, pp. 317–347.

Haav, H.-M., Ojamaa, A. (2017). Semi-automated integration of domain ontologies to DSL meta-models. *The International Journal of Intelligent Information and Database Systems (IJIIDS)*, 10(1-2), 94–116.

Haav, H.-M., Ojamaa, A., Grigorenko, P., Kotkas, V. (2015). Ontology-based integration of software artefacts for DSL development. In: Ciuciu, I., *et al.* (Eds.), *Proceedings of Confederated International Workshops of on the Move to Meaningful Internet Systems Conference*. *LNCS*, Vol. 9416. Springer, pp. 309–318.

Happel, H.J., Seedorf, S. (2006). Applications of ontologies in software engineering. In: *Proceedings of International Workshop on Semantic Web Enabled Software Engineering (SWESE'06) on the 5th International Semantic Web Conference, ISWC 2006*. *LNCS*, Vol. 4273. Springer, pp. 1–14.

Henderson-Sellers, B. (2011). Bridging metamodels and ontologies in software engineering. *The Journal of Systems and Software*, 84, 301–313.

Herre, H., Heller, B., Burek, P., Hoehndorf, R., Loebe, F., Michalek, H. (2006). *General Formal Ontology (GFO) – A Foundational Ontology Integrating Objects and Processes [Version 1.0]*. Onto-Med Report 8, Research Group of Ontologies in Medicine, Institute of Medical Informatics, Statistics and Epidemiology, University of Leipzig, Leipzig.

Hoehndorf, R., Prüfer, K., Backhaus, M., Herre, H., Kelso, J., Loebe, F., Visagie, J. (2006). A proposal for a gene functions wiki. In: *Proceedings of OTM2006 Workshops. Part I, Workshop of Knowledge Systems in Bioinformatics, KSinBIT 2006*. *LNCS*, Vol. 4277. Springer, pp. 669–678.

Hoehndorf, R., Loebe, F., Poli, R., Herre, H., Kelso, J. (2008). GFO-Bio: a biological core ontology. *Applied Ontology*, 3(4), 219–227.

Hoehndorf, R., Ngonga Ngomo, A.-C., Herre, H. (2009). Developing consistent and modular software models with ontologies. In: *Proceedings of the Eighth Conference on New Trends in Software Methodologies, Tools and Techniques*. IOS Press, pp. 399–412.

ISO, (1990). *IEEE Standards Association, Information Technology – Information Resource Dictionary System (IRDS)*. International Standard ISO/IEC 10027, ISO and IEC.

Kalnins, A., Barzdins, J., Celms, E. (2005). Model transformation language MOLA. In: *Proceedings of MDAFA 2003*. *LNCS*, Vol. 3599. Springer, pp. 62–76.

Kalnins, A., Kalnina, E., Celms, E., Sostaks, A. (2010). From requirements to code in a model driven way. In: *Proceedings of Advances in Databases and Information Systems (ADBIS) 2009*. *LNCS*, Vol. 5968. Springer, pp. 161–168.

Katasonov, A. (2012). Ontology-driven software engineering: beyond model checking and transformations. *International Journal of Semantic Computing*, 6(2), 205–242.

Katasonov, A., Palviainen, M. (2010). Towards ontology-driven development of applications for smart environments. In: *Proceedings of Workshops of IEEE International Conference on Pervasive Computing and Communications*. IEEE Press, pp. 696–701.

Kotkas, V., Ojamaa, A., Grigorenko, P., Maigre, R., Harf, M., Tyugu, E. (2011). CoCoViLa as a multifunctional simulation platform. In: *SIMUTools: Proceedings of ICST Conference on Simulation Tools and Techniques, ICST*, pp. 198–205.

Liuha, P., Lappetelainen, A., Soininen, J.-P. (2009). Smart objects for intelligent applications – first results made open. *ARTEMIS Magazine*, 5, 27–29.

Mints, G., Tyugu, E. (1982). Justification of the structural synthesis of programs. *Science of Computer Programming*, 2(3), 215–240.

Motik, B., Patel-Schneider, P.F., Horrocks, I. (2012). OWL 2 web ontology language: structural specification and functional-style syntax. http://www.w3.org/TR/owl2-syntax.

Ojamaa, A., Haav, H.-M., Penjam, J. (2015). Semi-automated generation of DSL meta models from formal domain ontologies. In: *MEDI: Proceedings of the Model and Data Engineering Conference*. *LNCS*, Vol. 9344. Springer, pp. 3–15.

OMG. (2002). *Meta Object Facility (MOF) Specification, Version 1.4*. OMG.

OMG. (2003a). MDA guide 1.0.1. Online http://www.omg.org/mda.

OMG. (2003b). *Ontology Definition Metamodel Request for Proposal*. OMG Document ad/2003-03-40 and http://www.omg.org/spec/ODM/.

OMG. (2006). Ontology Definition Metamodel RFP. Online http://www.omg.org/dontology/.

Palviainen, M., Katasonov, A. (2011). Model and ontology-based development of smart space applications. In: *Pervasive Computing and Communications Design and Deployment: Technologies, Trends, and Applications*. IGI Global, pp. 126–148.

Palviainen, M., Kuusijarvi, J., Ovaska, E. (2014). A semi-automatic end-user programming approach for smart space application development. *Pervasive and Mobile Computing*, 12, 17–36.

Pan, J., Staab, S., Assmann, U., Ebert, J., Zhao, Y. (2013). *Ontology-Driven Software Development*. Springer.

Parreiras, F.S., Staab, S. (2010). Using ontologies with UML class-based modeling: the TwoUse approach. *Data and Knowledge Engineering*, 69(11), 1194–1207.

Parreiras, F.S., Staab, S., Schenk, S., Winter, A. (2008). Model driven specification of ontology translations. In: *Proceedings of 27th International Conference on Conceptual Modeling*. *LNCS*, Vol. 5231. Springer, pp. 484–497.

Parreiras, F.S., Walter, T., Staab, S., Saathoff, C., Franz, T. (2009). Apis agogo: automatic generation of ontology apis. In: *Proceedings of the 3rd IEEE International Conference on Semantic Computing (ICSC 2009)*. IEEE Computer Society Press, pp. 342–348.

Ruy, F.B., de Almeida Falbo, R., Barcellos, M.P., Costa, S.D., Guizzardi, G. (2016). SEON: a software engineering ontology network. In: *Proceedings of EKAW' 2016*, pp. 527–542.

Souza, E.F., Falbo, R.A., Vijaykumar, N.L. (2013). Using ontology patterns for building a reference software testing ontology. In: *Proceedings of 17th IEEE International Enterprise Distributed Object Computing Conference Workshops*. IEEE Press, pp. 21–30.

Staab, S., Walter, T., Groner, G., Parreiras, F.S. (2010). Model driven engineering with ontology technologies. In: *Proceedings of Summer School on ReasoningWeb*. *LNCS*, Vol. 6325. Springer, pp. 62–98.

Sutii, A.M., Verhoeff, T., van den Brand, M.G.J. (2014). Ontologies in domain specific languages – a systematic literature review. *Computer Science Reports*, 14-09. Eindhoven. http://library.tue.nl/catalog/.

Tetlow, P., Pan, J., Oberle, D., Wallace E., Uschold, M., Kendall, E. (2006). Ontology driven architectures and potential uses of the semantic web in software engineering. W3C, Semantic Web Best Practices and Deployment Working Group, Draft online https://www.w3.org/2001/sw/BestPractices/SE/ODA/060103/.

Valaski, J., Reinehr, S., Malucelli, A. (2016). Which roles ontologies play on software requirements engineering? A systematic review. In: *Proceedings of International Conference of Software Engineering Research and Practice, (SERP'16)*. CSREA Press, pp. 24–30.

Walter, T., Ebert, J. (2009). Combining DSLs and ontologies using metamodel integration. In: Taha, W.M. (Ed.), *Domain-Specific Languages: IFIP TC 2 Working Conference, DSL 2009*. *LNCS*, Vol. 5658. Springer, pp. 148–169.

Walter, T., Parreiras, F.S., Staab, S., Eberet, J. (2010). Joint language and domain engineering. In: *Proceedings of 6th European Conference on Modelling Foundations and Applications*. *LNCS*, Vol. 6138. Springer, pp. 321–336.

Walter, T., Parreiras, F.S., Staab, S. (2014). An ontology-based framework for domain-specific modeling. *Software and System Modeling*, 13(1), 83–108.

H.-M. Haav is a senior researcher at the Department of Software Science of Tallinn University of Technology, Estonia. She received her PhD in Computer Science from the Institute of Cybernetics of the Estonian Academy of Sciences, Estonia. Her current research is focused on ontology engineering and ontology integration to model-based software development.