ABSTRACT

Conceptual models are built with concepts and relationships between them to reach a unified view of domain problems. There are many kinds of conceptual models developed in different modeling languages, such as class diagrams and entity-relationship models. In this paper, the authors have developed a specific meta-model following the Ecore standard to define conceptual models. These domain-specific conceptual models can be automatically formalized as domain ontologies using model transformation with the technique of triple graph grammars into ontology formal descriptions in accordance with the defined Ecore meta-model of the language OWL (web ontology language). For ontology deployment, its OWL code may be generated from OWL models using model-to-code transformation guided by Xpand templates. A performance evaluation is realized using a benchmark from the university domain with very large conceptual models. Through the experiments, they validate the performance and we prove the exactness and the scalability of the automatic transformation process of conceptual models.

KEYWORDS

Conceptual Model, Description Logic, MDA (Model-Driven Architecture), Model Transformation, Ontology, OWL (Web Ontology Language), Semantic Web, TGGs (Triple Graph Grammars)

1. INTRODUCTION

Conceptual models represent the static characteristics of a system, and they refer to models that have been formed after they have been conceptualized. Ontology is a formal description of a domain conceptualization (Proper & Guizzardi, 2021), which is composed of a set of names for concepts, roles and individuals to represent a knowledge base composed of a terminology set and a set of assertions. In the terminology part (TBOX), we define axioms for general concepts and role inclusions. The roles (relations) can be between pairs of concepts (called object properties) or between concepts and data types (called data type properties). The set of assertions (ABOX) are instances of concepts (concept

DOI: 10.4018/IJISMD.305229

This article published as an Open Access article distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/4.0/) which permits unrestricted use, distribution, and production in any medium, provided the author of the original work and original publication source are properly credited.
membership axioms) to assert that an individual (object) belongs to a concept, instances of object properties (object property membership axioms) to assert that an individual has a relation to another individual, or instances of data type properties (data type property membership axioms) to assert that an individual has a data property.

The main domain of ontology application is the Semantic Web, where Web documents are annotated with information (meta-data) from ontology terminology. These Web documents are instances of ontology concepts and roles. Thus, the annotation process is considered as the creation of assertions. The terminology and assertion sets represent the knowledge base of Web agents (programs). The Semantic Web has an architecture composed of resources (Web documents) and a set of intelligent agents. Each one has its own ontology.

As an alternative to keyword matching, ontologies are used in information retrieval as an intelligent search tool via an inference mechanism. The role of these intelligent agents is to answer user queries by running inference rules on their own knowledge base (ontology). Web intelligent agents can derive conclusions from shared knowledge bases linked by URI (Universal Resource Identifier) references. If an intelligent agent finds in its local query answer a reference to a distant knowledge base, it sends the URI reference to the appropriate agent. The former will dereference the URI reference to find its ontology and infer from its knowledge base to send back the result as a response. In this way, all the Web intelligent agents communicate to answer user queries using their proper ontologies (Bourahla, 2018).

An ontology describes Web resources as a graph-based model and it is formalized in the language OWL (Web Ontology Language), which is based on RDF (Resource Description Framework). The graph-based model can be serialized into a set of triples using one of several syntaxes, including XML, Turtle, and functional formats. A triple is composed of a subject, a predicate and an object. The subject is a Web resource (document), the object can be a Web resource (document) or a literal. The predicate is an OWL or RDF (RDFS) property (relation) that relates the subject to the object. The OWL/RDF model contains the ontology terminology and assertions as a set of RDF triples. Thus, with this OWL/RDF model, we can declare axioms, like concept inclusion axioms and membership axioms.

There are many technologies for developing ontologies that propose methodologies for developing ontology conceptualization, such as Ontolingua (Fikes et al., 1997), which was developed by Stanford University’s Knowledge System Laboratory. Ontolingua is devoted to ontology development using a form-based Web interface, and tools to formalize the conceptualization results, like Protégé (Tudorache et al., 2011).

In this paper, we propose a tool for modeling and formalization of ontologies. A graphical editor was developed to model ontology conceptualization with respect to the defined meta-model in the Ecore standard, using UML notations. Ecore (Steinberg et al., 2008, 2009) is the core meta-model of EMF. It allows the expression of other models by leveraging its constructs. The environment is defined in terms of itself (its own meta-model). The modeling result will be transformed using Triple Graph Grammars (Anjorin et al., 2015) into an ontology model with respect to a defined OWL2 ontology meta-model developed with Ecore. This bidirectional transformation can be realized by executing a set of rules defined with respect to a correspondence meta-model between the ontology-conceptual meta-model and the OWL2-based ontology meta-model.

The generated OWL2-based ontology model represents the result of ontology conceptualization and it can be formalized using a description language based on the syntax of RDF/XML with a specific format as a functional format. In this work, the transformation of the ontology model to text (code) is used to generate this formal description, which is developed using the Xpand technique (Xpand documentation, 2020). Considering the number of people familiar with UML notations, this solution will be a good approach to modeling ontology conceptualization for ordinary developers.

This paper is organized as follows. In Section 2, we present a set of related works. Preliminaries are presented in Section 3. Section 4 begins by presenting preliminaries for this work, and then we present in Section 4.1 the meta-models that are used for the generation of OWL2 models. This
generation is based on model-to-model transformation, which is explained in Section 4.1.3. Section 4.2 presents the model-to-code transformation for the generation of the OWL2 code. Implementation of this approach and its performance evaluation are presented in Section 5. In the end, conclusions and perspectives are presented in Section 6.

2. RELATED WORKS

The authors (Brockmans et al., 2006a) have presented a MOF (Meta Object Facility) based meta-model and a UML profile for OWL DL and OWL Full. They have also developed a transformation between OWL ontologies and UML models and vice versa, using a Model Driven Approach. They have implemented and validated their approach with the Visual Ontology Modeler and the Integrated Ontology Development Toolkit. The work of (Brockmans et al., 2006b) has presented a MOF compliant meta-model and UML profile for the Semantic Web Rule Language (SWRL) as an extension of their previous work. The validity of instances of this meta-model is ensured through OCL constraints.

In (Belghiat & Bourahla, 2012), the authors have discussed how to implement an application which makes a transformation from a UML class diagram to an OWL ontology based on graph transformation by using the tool AToM3. For this task, they have developed a meta-model for UML class diagrams, and a graph grammar composed of several rules, which enable them to transform all what is modeled in their AToM3 generated environment to OWL ontology.

The author (Bernaschina, 2017) has illustrated that Model Driven Development (MDD) requires model-to-model and/or model-to-text (code) transformations to produce application code from high-level descriptions. They have assumed that creating such transformations is in itself a complex task, which requires mastering meta-modeling, ad-hoc transformation languages, and custom development tools.

In (Minh & Quang, 2020), a set of rules was added for transforming UML class diagrams to ontologies, for the transformation of attributes with data types as classes, of attributes defined as the structure of other attributes, of associations with types as an association class, of the recursive association, and the transformation of aggregations with qualification.

The authors (Tiexin et al., 2020) have proposed in their paper a configurable semantic-based (semi) automatic conceptual model transformation methodology that tries to reuse existing conceptual models to generate new models. They try to improve the efficiency of the building process.

In (Wardhana et al., 2020), the research proposes a model that can automatically transform a SysML Requirement Diagram into an OWL file to extract knowledge contained in the previous diagrams. The transformation process makes use of a transformation rule and an algorithm that can be used to change a SysML Requirement Diagram into an OWL ontology file, where XML Metadata Interchange (XMI) serialization is used to represent the transformation.

The paper (Kchaou et al., 2021) suggests the addition of rules for transforming annotated BPMN (Business Process Management) models to ontologies by accounting for the semantics of the BPMN model, and providing for all business objects and activities. The transformations generate in addition the OWL2 graphical representation.

In (Chasseray et al., 2021), the authors presented a generic meta-model for the extraction of heterogeneous data. The meta-model has been designed with two objectives: the need for genericity regarding the source of collected pieces of knowledge and the intent to stick to a structure close to an ontological structure.

Compared to these related works, our approach works on the complete syntax of OWL2 and it has a more complete process by which we use the result of the UML-like ontology conceptualization to be formalized using a language of description based on RDF/XML syntax with a specific format as the functional format. All the steps of this formalization process are components integrated into the Eclipse modeling framework in the form of plugins.
3. PRELIMINARIES

Our contribution, which is explained in Sections 4 and 5, is based on the following concepts and definitions about modeling and model transformation. Abstraction of a system focusing on a specific viewpoint is described as a model with a modeling language to provide better understanding of and to reason about it. Models have been divided into different categories and they have different forms: there are mathematical models, graph models, text models, etc. In the context of MDA (Melor et al., 2004), models are categorized into four abstraction levels: meta-meta-model, meta-model, model, and system (subject). Depending on precision, models are divided into three levels, namely: conceptual models, specification models, and implementation models (Fowler et al., 1999).

As a model captures only a part of the complete system (subject), several models could be built to represent one system; one model for each point of the view of the system. In this case, we do model transformation, which is another pilot of model-driven approaches. Model transformation is a process (program), which contains a sequence of activities operating on models. The aim of model transformation is to generate target models based on source models (Kleppe et al., 2003).

According to the content conveyed in models, model transformations can be divided into two classes: model-to-model and model-to-text (code). There are two kinds of transformations (Mens & Gorp, 2006). The first kind of transformation is the horizontal transformation, which is a transformation where the source and target models reside at the same abstraction level. The horizontal transformation is divided into two types: endogenous and exogenous transformations. Endogenous transformations are transformations between models expressed in the same modeling language, for example, refactoring. Exogenous transformations are transformations between models expressed using different modeling languages. For example, migration.

The second kind of transformation is the vertical transformation, which is a transformation where the source and target models reside at different abstraction levels. A refinement is a typical example where an initial specification can be gradually refined with respect to refinement rules that add more concrete details to successive refinements until it arrives at a full-fledged implementation. Model transformation methods are defined by leveraging certain model transformation technologies, such as QVT (QVT/OMG, 2008).

A meta-model is a model that defines modeling rules (syntax, semantics and constraints); it is a model of a modeling language. Meta-models are usually described using UML Class Diagrams (OMG, 2005). It is built by leveraging the meta-modeling languages as Ecore (Steinberg et al., 2009), which is widely used and mature meta-modeling language, and it is a lightweight version of MOF (Meta Object Facility) (OMG, 2003). The Ecore meta-model is defined for the Eclipse Modeling Framework (EMF) (Budinsky et al., 2003), where model elements can be represented using XMI, which stands for XML Metadata Interchange (OMG, 2011, Calegari & Szasz, 2013).

TGG (Triple Graph Grammar) is another model transformation technology (Schürr, 1995), which refers to a special type of graph grammar, mainly for bidirectional model-to-model transformations, and it has a declarative language similar to the language QVT-Relations, which is a QVT declarative language (Greenyer & Kindler, 2010). The specialty of triple graph grammars is that their production rules consist of three sub-graphs, two of which represent the two models/graphs involved (source and target graph). The third sub-graph, the so-called correspondence graph, connects related graph parts from the source and target graphs and thus lies between the other two graphs.

The TGG defines three model transformation languages: source, target, and correspondence, all of which operate on models that conform to their environmental meta-models. The TGG standard integrates the OCL (Object Constraint Language) to provide constraint expressions on any MOF/Ecore model or meta-model (OMG, 2006). A TGG transformation represents a consistent relationship between the two models. It can be used to generate one of the models so that the set of models will be consistent.
4. FORMAL DESCRIPTION OF CONCEPTUAL MODELS

The objective of our contribution is to generate OWL2 code from domain-specific conceptual models of domain ontologies. For this automatic generation of formal descriptions of ontology-specific conceptual models, we have developed two model transformation processes. The first process (Section 4.1) is model-to-model transformation, which is a horizontal and endogenous transformation, where the TGG method is used for the model transformation. For this, we defined three Ecore meta-models for the ontology-specific conceptual models (the source models), OWL2 models (the target models) and correspondence models for mapping elements from the source and target models.

The second process (Section 4.2) is model-to-text (code) transformation, which is a vertical and exogenous transformation, where the Xpand method (Xpand Documentation, 2020), which is a language specialized in code generation based on EMF models, is used for the model transformation. For this, we use the OWL2 models produced by the first process with respect to its Ecore meta-model as the source of this model transformation. The target is the OWL2 code (text) with respect to the Xpand templates (representing its Xpand meta-model).

4.1 Generation of OWL2 Ontology Models

For this automatic generation of OWL2 ontology models, we use transformation of models based on Model Driven Architecture (MDA). We will define it using the Eclipse Modeling Framework (EMF), the Ecore domain-specific conceptual meta-model (Sub-section 4.1.1), the Ecore OWL2 meta-model (Sub-section 4.1.2) and the Ecore correspondence meta-model (Sub-section 4.1.3).

In Sub-section 4.1.4, we define the TGG rules (a transformation rule is a description of how one or more constructs in the source language can be transformed into one or more constructs in the target language (Kleppe et al., 2003)) for graph rewriting to do model-to-model transformation (or model consistency) using the TGG interpreter, with respect to the defined three meta-models. All these meta-models and the TGG rules are used to generate Java code within the environment of the Eclipse Modeling Framework (EMF) to edit conceptual models of domain ontologies and then generate their formal description in the language of OWL2.

4.1.1 Ontology-Conceptual Meta-Model

The source models of the model-to-model transformation should conform to the ontology-conceptual meta-model, which is defined as conforming to the Ecore standard. The environmental packages in this meta-model are divided into three categories. The first package contains one Ecore class (Figure 1) named “UMLDiagrams” to relate the second and the third Ecore packages by an extension relationship. This class indicates that the meta-model should contain one package for the class diagram and multiple Ecore packages for the object diagrams. A class diagram can be an extension of the existing class diagrams referenced by their URIs.

The second Ecore package contains a class diagram (Figure 2), which contains Ecore classes (at least one class) to define ontology conceptual concepts and associations to define conceptual

Figure 1. Ecore class “UMLDiagrams”
relationships between the concepts. The conceptual concepts and associations are sub-classes of the entity class “Entity” with the attributes “name”, “label” and “comment”. A conceptual concept is a class named “Class”, which is a specialization of the class “Entity”. An association is another class named “Association”, which is a generalization of two Ecore classes, the class association “ClassAssociation” and the data association “DataAssociation”.

A class association is defined to relate two concepts (“Class”). The first concept is its source (“AssociationSource”) and the second concept is its target (“ClassAssociationTarget”). A class association has the characteristic attributes of being functional, inverse functional, transitive, symmetric, asymmetric, reflexive and/or irreflexive. A data association with a unique attribute of being functional is used to define data properties for objects of various classes (concepts). In this case, the data association target (“DataAssociationTarget”) has a “DataType” attribute to provide values for the concept property, which can be of various types such as integers, strings, and so on.

A concept (“Class”) can be equivalent, subclass or disjoint with another concept that can be a simple concept which is previously declared or a complex expression of a class “Class”. An expression of a complex “Class” can be created using the operators of class complement, union and intersection and/or association restriction, which can be class association restriction or data association restriction. An association restriction can be an existential (some), universal (only) or (min, max, exactly)-cardinality restriction on an association with a qualification represented as a class expression. An operator of an Ecore class expression can be unary with one operand (class) or binary with two operands (left and right Ecore class expressions). We can declare associations as equivalent, super, sub, inverse and disjoint associations with another association (see Figure 2 for more details).

The third Ecore package contains a class diagram named “ObjectDiagram” (Figure 3), which contains Ecore classes to define instances of the classes of concepts and associations. The conceptual model can have multiple object diagrams to model multi-source data sets. A class diagram of type “ObjectDiagram” is composed of a set of object entities. Each one has a name and it may have a label and/or a comment. A class (concept) can be represented by an object. It can have instances
of class association with other objects and it can have instances of data association with data of a specified data type.

Two object instances can be specified to be the same (same instance) or different (different instance). It is also possible to say that an object is not an instance of a class association (negative instance of class association). By contrast, we can say it is not an instance of a data association (negative instance of data association).

For example, Figure 4 describes, with respect to this conceptual meta-model, a model of ontology in the domain of family relations. It contains the names “Person”, “Parent”, “Mother” and “Father” for concepts (Ecore classes), where “Parent”, “Mother” and “Father” are sub-classes of the class “Person” and the class “Parent” is the union of the classes “Mother” and “Father”.

The class “Mother” is disjoint from the class “Father”. The names “hasChild” and “hasParent” are defined for class associations between the class “Person” and itself. The names “hasName” and “hasAge” are defined for data associations between the class “Person” and a data type (String and integer, respectively).

The object diagram contains the objects “Alice” (instance of the class “Mother”), “David” (instance of the class “Father”), “Bob” and “Bobby” (instances of the class “Person”), and they are all the same (“SameAs”). We can define “Bob” as being 16 years old (instance of the data association “hasAge”). “David” has a child, “Bob” (instance of the class association “hasChild”) and “Bob” has a parent, “David” (instance of the class association “hasParent”). “Alice” is the mother of “Bob” (instance of the class association “hasChild”).

4.1.2 OWL2-Ontology Meta-Model

OWL2 is a Semantic Web language based on description logic (SROIQ (D)) (Boris et al., 2012), designed by the W3C for expressing rich and complex knowledge about things, groups of things, and
relations between things. It is a computational logic-based language such that knowledge expressed in OWL2 can be reasoned with by computer programs either to verify the consistency of that knowledge or to make implicit knowledge explicit. OWL2 documents, known as ontologies, can be published on the World Wide Web and may refer to or be referred to other OWL2 ontologies.

In this subsection, we define an Ecore meta-model for OWL2, where the elements in the conceptual meta-model can be mapped to elements in the OWL2 meta-model. Then, models respecting their syntax and semantics can be used to generate an OWL2 code. The goal is to make users familiar with it to cope with the conceptualization of domain ontology and the transformation program will generate its OWL2 code. Thus, the definition of this Ecore meta-model is based on the OWL2 syntax and the structure of the Ecore conceptual meta-model.

The OWL2 meta-model is composed of three Ecore packages. The first package contains one Ecore class called “Ontology” (Figure 5) and it has three attributes: the first attribute is to import external ontologies by specifying their URIs, the second attribute is a reference to the second Ecore package, which contains the terminology and the third attribute is a reference to a set of assertion sets. Thus, this package can be considered as an extension of a TBOX package and multiple ABOX packages, which are explained below.

The second Ecore package (Figure 6) is composed of Ecore classes to define the ontology terminology (TBOX). The TBOX Ecore class is composed of classes to map elements from the conceptual meta-model with additional information required for the OWL2 code generation.

The TBOX class is composed of “Concept” and “Property” (generalization of “ObjectProperty” and “DataProperty”) classes, which are specializations of the class “Entity”. A “Concept” class can be equivalent, sub-class or disjoint with another “Concept” class that can be a concept complex expression. A property can have attributes representing some of its characteristics and it can be specified as an equivalent property, a sub-property, etc.

The OWL2 meta-model can also contain a set of Ecore packages representing boxes of assertions (ABOX) as multi-source data sets to define instances of concepts and properties (Figure 7). The main Ecore class is the class “Individual” to create individuals (objects) as members of concepts. Individuals can be related by object properties (instances of “ObjectProperty”) and they can have data properties (instances of “DataProperty”). We can negate these instances by declaring instances of “NegativeObjectPropertyAssertion” and “NegativeDataPropertyAssertion”, respectively. Two individuals can be declared to be the same with “SameAs” or different with “DifferentFrom”.

For example, the same model in Figure 4 can be described in the OWL2 meta-model as follows.

4.1.3 Correspondence Meta-Model for Model-to-Model Transformation

Model transformation is a process of generating a target model based on a source model; the transforming rules should be built between similar concepts that are from the two models, respectively. Triple Graph Grammars (TGGs) is a formally well-defined technique for model-to-model transformation. It is a declarative and rule-based bidirectional transformation (Anjorin et al., 2015). The general idea of TGGs is to define a language of integrated models, which consists of a

![Figure 5. Ecore class “Ontology” of the OWL2 meta-model](image)
Figure 6. Ecore class diagram “TBOX” of the OWL2 meta-model

Figure 7. Ecore class diagram “ABOX” of the OWL2 meta-model

Figure 8. Example of an Ecore OWL2 model for family relations
model of the source domain, a model of the target domain, and explicit correspondence (mapping) structures in the middle component.

The result of the TGG transformation consists of a set of rules that describe how a set of triples of source domain (ontology conceptualization), correspondence and target domain (OWL2) models can be produced simultaneously and can be viewed as a consistency relation. If we denote the models by \( G_X \), where \( G \) stands for graph and \( X \in \{ S, C, T \} \) represents the domain (source, correspondence, or target) of the model. Triples of models are consequently denoted by \( G_S \rightarrow G_C \rightarrow G_T \). The rules of a TGG generate a language of consistent triples, denoted by \( \mathcal{L}(TGG) \). With this notation, we can now formulate TGG-based consistency as:

\[
G_S \text{ is consistent with } G_T \text{ if and only if } \exists \subseteq \mathcal{L}(TGG) \text{ (Hildebrandt et al., 2013).}
\]

In other words the triple \( G_S \rightarrow G_C \rightarrow G_T \) can be generated with the TGG.

With TGGs, to guarantee that a source model is consistent with a target model, we specify a correspondence model between them, which will be used by a set of TGGs rules for checking the consistency. This correspondence model should conform to a defined correspondence meta-model. Also, the source and target models conform to their corresponding meta-models that are connected by the correspondence meta-model. The triple of source, target and correspondence meta-models is referred to as a TGGs schema.

In this context, we define the Ecore correspondence meta-model (Figure 9) for the model-to-model transformation (ontology conceptual model to OWL2 model), which creates elements of the target model. It can be viewed as a set of traceability links between corresponding model elements from the source and target domains.

The correspondence meta-model is composed of thirteen Ecore classes, which are all specializations of one Ecore class (the root class). Each specialized class is defined to map elements from the source and target models. For example, the Ecore class “Class2Concept” will be used to map the “Class” element from the source model to the “Concept” element from the target model.

Principally, a triple graph grammar is considered as a regular graph grammar with a set of graph grammar rules, where the empty graph is the axiom. These rules generate a language of graphs or, more precisely, the set of all consistent graphs, which is a subset of the set of all schema-compliant graphs. The interesting point of the TGG is the fact that the specified rules consist of three sub-rules and the generated graphs consist of three related sub-graphs.

Figure 9. Correspondence meta-model
A source component with a target component represents a pair of related graphs and the middle (correspondence) component introduces traceability relationships between the pair of graphs. The transformation process uses TGGs to generate a set of regular operational graph transformation rules tailored for a specific purpose, like “forward” model transformation or “backward” propagation of changes (König et al., 2018).

Figure 10 describes the TGGs principle, where elements from the model in the source domain are mapped to elements from the model in the target domain using the correspondence model. The general idea of defining the transformation rules is to edit the TGG rules as transformation resources.

To start the transformation, a starting node in the input model and an operational rule to translate this node must be determined (Figure 11). Such a valid starting rule without any context elements, i.e., no preconditions, is referred to as an axiom.

In the axiom, the ontology (output) model has the same name, label and comment as the conceptual model (“UMLDiagrams”). After the first node is translated, a strategy is required to systematically cover all elements in the input model in an appropriate order. The rule in (Figure 12) will create the target context of the ontology terminology (“TBOX”) from the source context named “ClassDiagram”. There are seventeen TGG rules developed for mapping elements for the context of TBOX.

Consequently, each class, class association and data association in the source context will be translated into concept, object property and data property, respectively in the target context. An equivalent, sub or disjoint class (in the context of the conceptual model) or concept (in the context of the TBOX model) can be expressed with complex expressions. A class (concept) expression can be a restriction on a class association (object property), or a data association (data property). A restriction may have a qualification, which is a class (concept) expression. A set of TGG rules are defined to map the Ecore elements of these class expressions.

Figure 10. TGGs Principle (König et al., 2018)

Figure 11. The TGG axiom
The mapping between elements in the contexts containing instances (objects) in the conceptual model (Ecore package of “ObjectDiagram”) and the OWL2 model (Ecore meta-model of the package “ABOX”) is defined with 9 TGG rules. The TGG rule in Figure 13 is defined to create the context of the assertion box “ABOX” from the instance context of the conceptual model.

In this context, we can produce individual elements “Individual” from the object element, types (classes) of created individuals, instances of object properties and data properties from instances of class and data associations, respectively. There are TGG rules to produce mapping between elements for the negation instances of object and data properties, to produce assertions about whether two objects (individuals) are the same or different.

Each TGG rule is saved in a file as a rule resource. All these resources are referenced by rule names created in a TGG file to be used for model-to-model transformation. For example, after validation of the correspondence meta-model with respect to the Ecore meta-model and validation of the set of TGG rules with respect to the TGG language, we can translate the model in Figure 4 to the model in Figure 8 using the TGG interpreter.

For that, we first create the TGG interpreter configuration by selecting the files containing the source model, the TGG rules, and the three meta-models (source, correspondence and target). In the second step, we perform the TGG transformation by selecting the created configuration file. A target model is generated as an XMI file that can be shown with the Ecore modeling editor. The TGG interpreter gives us the number of applied rules (54 TGG rules, for this example) and a transformation time is computed (105 ms, for this example).

Figure 12. Creating the context “TBOX”

![Figure 12. Creating the context “TBOX”]

Figure 13. Creation of the context “ABOX”

![Figure 13. Creation of the context “ABOX”]
4.2 Generation of OWL2 Code

The code is typically generated using templates, a technique that is extremely well established (for example in web development). A template can be thought of as the target code with holes for variable parts. The holes contain meta-code (so code creating code), which is run at template instantiation time to compute the variable parts. A template is written in the template language of Xpand (Xpand Documentation, 2020).

A template should import the meta-models of OWL, TBOXModel and ABOXModel defined in Section 4.1.2 and add some constraint checks. So, an input model for the code generator will have a structure conforming to these meta-models. The generated code respects the functional format of the RDF/XML syntax. Elements in the source model map to arbitrary fragments of the code.

The main template module, which is presented in Figure 14, will create an ontology file whose name is the name of the ontology given in the conceptual model with the prefix “.owl”. The generator writes in this file the ontology headers, and then it calls the template modules for expanding ontology declarations, class axioms, object axioms, data axioms and named individuals.

The expansion of the ontology declarations can be realized with respect to the template module named “declarations”. We have declarations of classes (concepts), object properties and data properties, which are declared in the TBOX model. The class (concept) axioms are generated with the template module named “class_axioms”. For each concept in the TBOX model, we have the axioms: labels, comments, annotations, disjoint with, equivalent to and sub concept elements.

Figure 14. The main Xpand template

```
<IMPORT oWL>
<IMPORT TBOXModel>
<IMPORT ABOXModel>
<EXTENSION template::GeneratorExtensions>

<DEFINE main FOR Ontology>
<FILE name=".owl">
Prefix(:=<http://www.ontology.org/<name>.owl#>)
Prefix(owl:=http://www.w3.org/2002/07/owl#)
Prefix(rdf:=http://www.w3.org/1999/02/22-rdf-syntax-ns#)
Prefix(xsd:=http://www.w3.org/2001/XMLSchema#)
Prefix(rdfs:=http://www.w3.org/2000/01/rdf-schema#)

Ontology(http://www.ontology.org/<name>.owl>
<IF label != null>"Annotation(rdfs:label "+ ""+label+""")"»ENDIF>
<IF comment != null>"Annotation(rdfs:comment "+ ""+comment+""")"»ENDIF>

<EXPAND declarations FOR this>
<EXPAND class_axioms FOR this>
<EXPAND object_axioms FOR this>
<EXPAND data_axioms FOR this>
<EXPAND named_individuals FOR this>
<ENDFILE>
<ENDDEFINE>
```
OWL2 code for object property axioms defined in the TBOX model is generated with the expansion module named “objet_axioms”. It contains expansions for domain and range axioms, characteristic (functional, inverse functional, symmetric, asymmetric, reflexive, irreflexive) axioms, and axioms for equivalence, inclusion, disjointness and super. In the same way, the Xpand module “data_axioms” is used to produce the OWL2 code for TBOX axioms about the different ontology data properties.

The expansions to the assertion axioms defined in the ABOX model are realized by the Xpand module “named_individuals”. There are expansions in assertions about individual types, object (data) properties as relations between two individuals (individual and data), negation of assertions about object (data) properties, and assertions about whether individuals are the same or different. In addition to these template modules, an Xpand module named “concept_expression” is defined to expand concept expressions.

The second transformation is a model-to-code transformation, which takes an OWL2 model and generates a code in the OWL2/XML functional format. This step can be accomplished by calling the Xpand template from a workflow.

After executing the workflow, the generated code should be acceptable in the OWL2 language.

5. IMPLEMENTATION AND PERFORMANCE EVALUATION

The implementation combines two main tasks: model-to-model transformation and model-to-text (code) transformation. Model-to-model transformation consists of defining a meta-model for ontology conceptualization, a meta-model for OWL2/XML-based ontology description, a meta-model for correspondence between source and destination meta-models, and a transformation grammar based on TGGs to translate the ontology conceptual model to an OWL2/XML description model. Model-to-code transformation consists of defining the Xpand template with respect to the OWL2 meta-model and the RDF/XML syntax with respect to the functional format.

The Eclipse project for model-to-model transformation was developed to integrate all these meta-models with the TGG rules to realize the generation of target models from the edited source models. The EMF (Eclipse Modeling Framework) ECore (Steinberg et al., 2008) of the Eclipse platform is used for the implementation of this automatic transformation. The meta-model (Ecore) of EMF Eclipse is used to define models and it provides run-time support for handling the models of the core EMF framework, so as to generate interfaces to edit objects, which separate application development from class implementation.

Thus, the meta-modeling layer of the EMF tool is used to graphically model the meta-models of the conceptual (class and objet) diagrams, which represent the source meta-model of ontology conceptualization (“UMLdiagrams”) and the target meta-model of the OWL2 ontology (“TBOX” and “ABOX”). The correspondence meta-model is added to connect the source and the target meta-models. After validation of all the meta-models with respect to the syntax of the EMF Ecore meta-model, the Java code is generated from the EMF Generator Model to complete the project implementation for the model and editor codes.

The TGG Interpreter is used to generate OWL2-based ontology models, by executing graph rewriting rules based on the source, target and correspondence meta-models. These rules are specified (see Section 4.1.3.2) to do model-to-model (M2M) transformations using the TGG technique. These transformations are realized between graphs modeled on the Eclipse Modeling Framework (EMF). Thus, to generate the OWL2-based ontology models with the TGG interpreter, we begin by writing transformation (TGG) rules using TGG, which are sensitive to the defined meta-models. Then, write statements with the Object Constraint Language (OCL) to assign values to specific constraints and, after rules validation, the TGG rules will be boosted by the relations of generalization.
The Eclipse project for model-to-code transformation (Xpand generator) integrates the Xpand template (Section 4.2) and the OWL2 meta-model (Section 4.1.2) to generate an OWL2/XML formal description of the ontology from its OWL2 model. Xpand is a template engine for generating code from EMF models (Xpand Documentation, 2020). It uses the modeling workflow engine (MWE) to execute different Eclipse modeling components within Eclipse as well as standalone using configured workflows with a declarative XML-based language (Eclipse Modeling M2T, 2020).

The Xpand project provides the runtime to execute workflows and the IDE tooling to edit them. The Xpand template is a target code containing holes as variable parts. These holes have a meta-code, which means that the code creates a code. The variable parts are computed during the template instantiation time. In this case, we use the Xpand language to create templates (see template code in Section 4.2) and add checking constraints for code generation of OWL2 code from EMF Ecore models of the OWL2 ontology generated by the TGG interpreter using the TGG rules for model-to-model transformation of ontology conceptual models.

5.1 Evaluation Metrics and Test Data

The performance evaluation of this formalization process of ontology conceptualization is based on evaluation of its scalability on Semantic Web data with large sizes, which should commit to semantically rich ontologies and of its exactness. Therefore, the evaluation will measure the tradeoffs between scalability and formalization exactness.

The performance evaluation is based on a set of metrics. The first metric is the number of applied TGG rules to do model-to-model transformation. The second metric is the time spent on this transformation (the sum of times spent on model-to-model and model-to-code transformations) and the third metric is the transformation exactness (its completeness and soundness).

A transformation is complete if it generates all (correct) entities composing the ontology. The degree of completeness of a transformation is measured as the percentage of the generated ontology entities. The percentage of ontology entities generated by the transformation that actually composes the ontology is used to determine the degree of soundness of a transformation. For example, if the ontology is composed of N entities (according to the benchmark are all correct) and the transformation has generated M entities (correct and not correct), if we have K correct entities from the M generated entities then the completeness is \((K/N)\times100\%\) and the soundness is \((1-((M-K)/M))\times100\%\).

In addition to these metrics, there is a combined metric for formalization performance that measures formalization time and exactness in combination to aid in evaluating the potential tradeoff between formalization time and formalization capability, as well as the overall performance of conceptualization formalization.

To measure these performance metrics, many experiments should be done on popular and well-established ontologies, which have been used in previous benchmarks. The chosen benchmark is the Lehigh University Benchmark (LUBM) (Guo et al., 2005), which has an ontology about the university domain and test data (ABoxes). In order to evaluate the ability of the ontology-conceptualization formalization to handle large ABoxes, we need to be able to vary the size of the data (assertions about instances), and see how it scales.

The university ontology is expressed in OWL Lite, which is a sublanguage of OWL2. The ontology describes 43 concepts, 32 properties composed of 25 object properties and 7 data type properties. It uses OWL Lite language features including inverse, transitive property, some values from (existential) restrictions, and intersection of, where 263 axioms are described, which are composed of 93 logical axioms, 44 concept inclusion axioms, 5 property inclusion axioms, 2 inverse axioms, 1 transitive axiom, 25 domain axioms, 18 range axioms and 75 annotation assertion axioms.

A generator named UBA (Univ-Bench Artificial data generator), was developed to generate test data, which can be scaled to an arbitrary size, where a university is the minimum unit of data generation. For each university, a set of OWL files describing its departments is generated. There
are parameters for the generator UBA to specify how many and which universities to generate. The OWL Lite also describes the test data generated by the generator UBA. Each ontology uses different datasets with an increasing ABox size. In a LUBM dataset, every university contains 15 to 25 departments; each is described by a separate OWL file.

We have created 9 sets of test data $DS^{Org}_k$, which contain OWL files for $k$ universities. Each dataset $DS^{Org}_k$ has a number of departments from 15 to 25, where each university department is described by an OWL file. These data sets are input to another tool, which we developed to generate Ecore conceptual models with respect to the conceptual meta-model, and they will be effectively used to evaluate performance. We create a conceptual Ecore model $DS^{Con}_k$, for each original test data set $DS^{Org}_k$, which is then used as input to the combined model-to-model and model-to-code transformations that generate the data set $DS^{Gen}_k$. The exactness is measured by computing the difference between them.

### 5.2 Experimental Results for Performance Evaluation

The tests were performed on a laptop with a 1.70 GHz Intel (R) Core (TM) i5-4210U processor and 8 GB of RAM using Windows 10 (64 bits). The experiments are to see if we can handle these large-scale datasets. Table 1 shows the sizes in mega-bytes, the number of applied rules and the transformation time of each dataset. The results show that the tool was able to process a university with 24 departments in 176.52 minutes. However, for 25 departments, it spent more than 180 minutes (3 hours), which is the time limit. Thus, it displays good scalability in data transformation compared to ontology loading tested with the same benchmark (Guo et al., 2005). This high scalability is due to space savings in the transformation process and the way the grammar rules are rewritten.

Table 1 shows the comparison of dataset completeness to see what benchmark test ontology can be generated by the transformation procedure and then test its transformation capability. The results show that the transformation can generate almost all ontology entities (almost all entities in $DS^{Org}_k$ are in $DS^{Gen}_k$). Also, Table 1 shows the soundness of the transformation procedure for each dataset. It was discovered that it transforms all datasets almost perfectly (all entities in $DS^{Gen}_k$ are in $DS^{Org}_k$).

<table>
<thead>
<tr>
<th>#</th>
<th>$S(DS^{Org}_k)$ (MB)</th>
<th>$S(DS^{Con}_k)$ (MB)</th>
<th>#Applied TGG Rules</th>
<th>Generation Time (mn)</th>
<th>$S(DS^{Gen}_k)$ (MB)</th>
<th>Completeness ($C_{DS^{Con}_k}$)</th>
<th>Soundness ($S_{DS^{Con}_k}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8.43</td>
<td>8.09</td>
<td>103481</td>
<td>1.02</td>
<td>8.42</td>
<td>100%</td>
<td>99%</td>
</tr>
<tr>
<td>2</td>
<td>19.39</td>
<td>18.78</td>
<td>237524</td>
<td>2.33</td>
<td>19.35</td>
<td>98%</td>
<td>97%</td>
</tr>
<tr>
<td>3</td>
<td>28.44</td>
<td>26.95</td>
<td>348417</td>
<td>8.62</td>
<td>28.39</td>
<td>100%</td>
<td>91%</td>
</tr>
<tr>
<td>4</td>
<td>40.37</td>
<td>35.74</td>
<td>494152</td>
<td>20.81</td>
<td>40.31</td>
<td>95%</td>
<td>91%</td>
</tr>
<tr>
<td>5</td>
<td>52.80</td>
<td>47.81</td>
<td>646143</td>
<td>74.78</td>
<td>52.74</td>
<td>96%</td>
<td>92%</td>
</tr>
<tr>
<td>6</td>
<td>61.11</td>
<td>54.92</td>
<td>748020</td>
<td>108.28</td>
<td>60.94</td>
<td>96%</td>
<td>89%</td>
</tr>
<tr>
<td>7</td>
<td>74.76</td>
<td>64.73</td>
<td>914826</td>
<td>149.93</td>
<td>73.98</td>
<td>93%</td>
<td>92%</td>
</tr>
<tr>
<td>8</td>
<td>84.67</td>
<td>71.52</td>
<td>1036201</td>
<td>176.52</td>
<td>82.87</td>
<td>91%</td>
<td>88%</td>
</tr>
</tbody>
</table>
Through investigation of these results, we found that the original ontologies contained information which was not taken into account because it was initially considered to be useless information. For this reason, the formalization is not completely complete and sound.

From Table 1, the tool shows that there was a proportional increase in the number of applied rules and the transformation time as the data size grows. It is important to evaluate the overall performance of the tool by combining the above metric results. The metric $F_{DS^\text{Con}}$ is used to compute the tradeoff between transformation completeness and soundness, since, essentially, they are analogous to recall and precision in information retrieval, respectively.

$$F_{DS^\text{Con}} = \frac{(\beta^2 + 1) \times C_{DS^\text{Con}} \times S_{DS^\text{Con}}}{\beta^2 \times C_{DS^\text{Con}} + S_{DS^\text{Con}}}$$

In the preceding formula, $C_{DS^\text{Con}}$ and $S_{DS^\text{Con}}$ ($\in [0,1]$) represent the dataset’s ($DS^\text{Con}_k$) transformation completeness and soundness, respectively. Thus, this formula is used to determine the relative weighting between them. In addition to calculating the transformation performance, we use the metric $P_{DS^\text{Con}}$ for the dataset $DS^\text{Con}_k$.

$$P_{DS^\text{Con}} = \frac{1}{1 + e^\gamma \times T_{DS^\text{Con}} / N_k - \delta}$$

In the formula, $T_{DS^\text{Con}}$ is the transformation time $T$ (ms) for the dataset $DS^\text{Con}_k$ (sum of times spent by model-to-model and model-to-code transformation). The number $N_k$ is the total number of applied TGG rules to transform the conceptual model of the system ($DS^\text{Con}_k$). To allow for comparison of the metric values across datasets of different sizes, we use the transformation time per TGG rule (i.e. $T_{DS^\text{Con}} / N_k$) in the calculation. To distinguish between the experiment results over different datasets, two control parameters $\gamma$ (called the slope) and $\delta$ (called the shift) are used. Their values are chosen to make $P_{DS^\text{Con}}$ (the fast transformation with respect to $N_k$) close enough to one (above 0.99). For these experiments, $\gamma = 2$ and $\delta = 20$.

The overall performance metric $OP$ rewards the transformation procedure when it produces entities faster, more completely and more soundly and it is defined as a composite metric of transformation time, completeness and soundness for each dataset $DS^\text{Con}_k$, where $k = 1,\ldots,8$ as in the following.

$$OP_{DS^\text{Con}} = w_{DS^\text{Con}} \times \frac{(\alpha^2 + 1) \times P_{DS^\text{Con}} \times F_{DS^\text{Con}}}{\alpha^2 \times P_{DS^\text{Con}} + F_{DS^\text{Con}}}$$

where $D$ is the total number of test datasets ($D = 8$), $w_{DS^\text{Con}}$ ($\sum_{i=1}^{D} w_{DS^\text{Con}} = 1$) is the weight given to dataset $DS^\text{Con}_k$ which determines relative weighting between $F_{DS^\text{Con}}$ and $P_{DS^\text{Con}}$.

Figure 15 shows the results of the overall performance metric $OP$ of the transformation procedure that is calculated with respect to each dataset, where both $\beta$ and $\alpha$ values are set to 1.
which means the weights of transformation completeness and transformation soundness are equal, and the weight of transformation time against the weight of completeness and soundness are also equal. The weight $w_{DS_{k}^{Con}}$ of $DS_{k}^{Con}$ is calculated by the formula $w_{DS_{k}^{Con}} = N_{k} / \sum_{j=1}^{D} N_{j}$, where $k \in \{1, \ldots, D\}$ and $N_{k}$ is the total applied TGG rules for transforming the dataset $DS_{k}^{Con}$.

Figure 15 shows that the tradeoff between completeness and soundness of the transformation, is more than 90%, with transformation performance being 100% for the first five data sets and decreasing as the data sets get larger. The overall performance indicates that the transformation process performs very well for the data set number 6 compared to the other datasets. These numerical results, shown in Figure 15, help us understand the overall transformation performance of our approach.

6. CONCLUSIONS AND PERSPECTIVES

In this work, we have proposed an approach for automatic formalization of ontology-specific conceptualization. The ontology conceptualization is edited as a model with respect to the meta-model defined by the EMF Ecore meta-modeling language. The conceptual model will be first transformed using the TGG technique to an OWL2 model, where a target OWL2 meta-model and a correspondence meta-model are defined with the EMF Ecore meta-model language to use the TGG transformation technique. This model-to-model transformation uses a set of TGG rules defined by the TGG interpreter language.

Then, the produced OWL model will be transformed to an RDF/XML functional format using model-to-code transformation with the technique of Xpand/MWE. This approach was extensively tested with a known benchmark for the university domain, which contains very large datasets. The results show its scale and prove its exactness. The goal of this work is to extend it to the Semantic Web Rule Language (SWRL) and use it to assist users in developing ontology conceptualizations based on reusing enterprise information systems.
CONFLICT OF INTEREST

The authors of this publication declare there is no conflict of interest.

FUNDING AGENCY

The publisher has waived the Open Access Processing fee for this article.
REFERENCES


Malika Boudia is PhD student in Computer Science Department at the University of M’Sila. She is working on ontologies and model transformations.

Mustapha Bourahla is a professor in Computer Science Department at the University of M’Sila (Algeria). He is working on formal methods and ontologies.