Towards an Ontology Engineering Framework for Integrating Visualisation, Metamodelling and Reasoning

Emiliano Rios Gavagnin[1], **Germán Braun**[1,3], **Laura Cecchi**[1], **Pablo Fillottrani**[2,4]

[1]Universidad Nacional del Comahue, Argentina

[2]Universidad Nacional del Sur, Argentina

[3]Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET)

[4]Comisión de Investigaciones Científicas de la Provincia de Buenos Aires (CIC)

`emiliano.rios@est.fi.uncoma.edu.ar`

`{german.braun,lcecchi}@fi.uncoma.edu.ar,prf@cs.uns.edu.ar`

Abstract. *Conceptual data modelling languages are a common approach to represent ontologies visually. This paper introduces a framework aiming at enabling modelling tools to represent, manipulate and reason over OWL ontologies represented in different visual languages, such as UML, EER or ORM 2, based on a previously developed theoretical metamodel. We describe the framework and define a methodology to exploit it, in order to make visual independence a reality. Finally, we present case studies on realistic ontologies to show the applicability of our approach.*

1. Introduction

Visualisations provide suitable abstraction levels for ontologies and their techniques help users to enhance the interaction with the tools by encoding information in a manipulable medium. However, both the lack of widely accepted methods [Dudáš et al. 2018] in addition to the fragmented methodologies and tools in the landscape of Semantic Web technologies [Vigo et al. 2014], have grounded the need of defining suitable and flexible system requirements.

Conceptual data modelling languages (CDMLs) are a common approach to represent OWL ontologies visually [Fillottrani et al. 2012, Cerans et al. 2012]. Normally, each modeller has their own preferences about which conceptual modelling language to use during the data analysis and design stages. Moreover, stakeholders may have different level of knowledge, understanding and cognitive capability for comprehending models [Ong and Jabbari 2019]. Thus, to enhance the collaboration among them, an important task is to establish the relation that underlies these conceptual modelling languages to provide interoperability, integration and conversion of conceptual models represented in such languages [Keet and Fillottrani 2013].

On the other hand, the development of high-quality models needs to include automated reasoning technologies to ensure their correctness. Giving tools the ability to detect errors and unexpected (non-) entailments [Horrocks 2011, Glimm and Stuckenschmidt 2016] is key for assuring certain quality parameters and adequate development times and costs.

Aiming to put together these related aspects, we have previously identified the following set of system requirements framed into a web architecture for visual ontology engineering tasks, named crowd[1] [Braun et al. 2020]: *(R1) Conceptual Modelling Support*, *(R2) Visual Editing*, *(R3) Integrated Reasoning Services*, *(R4) Visual Language Independence*, *(R5) Simultaneous Manipulation of Visual Models*, *(R6) Extensible Framework* and *(R7) Semantic Web Compatibility*. Some of them, R1, R2, R6 and R7, have been fulfilled by crowd [Braun et al. 2020], so that in this work, we focus specifically on three of them: *R3*, *R4* and *R5*, where we consider the separation of concerns into formal features and visual functionalities. Thus, we take metamodelling aspects as formal ones, which give independence of how the models are (visually) represented, while addressing aspects related to the vital support of automated reasoning.

(R4) Visual Language Independence and (R5) Simultaneous Manipulation of Visual Models The independence of visual languages enables tools to represent and manipulate the very same ontology in different visual languages, according to modellers' preferences. The variety of CDMLs is wide, and preferences may vary according to end users or modellers, communities and research fields. Therefore, achieving language independence is a key factor when creating an ontology design tool. In this sense, the Keet-Fillottrani (KF) Metamodel [Keet and Fillottrani 2013] is a unifying ontology-driven metamodel of the three main language families, EER, UML and ORM 2 conceptual data modelling languages, inclusive of all constructs of the languages under consideration. This metamodel has been completely formalised in FOL [Fillottrani and Keet 2014b], and thus ensuring the precision of meaning and determining its expressiveness and complexity.

(R3) Integrated Reasoning Services Visual aspects can help modellers detecting anti-patterns and debugging models, however, the increasing complexity of models requires a formal encoding and automated reasoning to ensure their quality. Nowadays, reasoning systems are integrated into a huge range of tools but some interoperability issues and undetected deductions are still present [Braun et al. 2019a]. On the other hand, the use of Description Logics (DLs) to define precisely the semantic of UML, EER and ORM 2 has been widely studied [Berardi et al. 2003, Artale et al. 2007, Franconi et al. 2012]. Therefore, the aim of integrating automated reasoning with visual aspects is to take advantage of them to manage both the original model and its (possible) implicit deductions in a graphical way.

In summary, suitable visualisations, automated reasoning and metamodelling are key features in ontology engineering to facilitate the analysis and design of models and successfully integrate modeller's intentions with the formal semantics. With this work, we go one step further towards satisfying these requirements. Our main contributions are:

1. the development of a framework that integrates visual editing over EER, UML and ORM 2 conceptual modelling languages, which covers R4 and R5 requirements. This capability is based on the solid theoretical background of the KF metamodel. Here, we implement the subset of interoperability rules, given by KF, which unifies the commonalities of the selected CDM languages.
2. the development of a new interface for partial visualisation of OWL ontologies in any of the supported notations, by mapping OWL axioms to KF, integrated on the

[1]`http://crowd.fi.uncoma.edu.ar/`

framework. This feature aims to assist people who are not familiar with OWL but are familiar with some other notation such as UML, EER, or ORM 2.

3. the design of a new version of crowd core implementing the encoding of KF metamodel instances as DL knowledge bases, which satisfies the R3 requirement.
4. a methodology, where we exploit visualisation, metamodelling and reasoning for dealing with ontology development tasks. The methodology makes use of custom visualisations but keeping only one common and unified model. It enables representing ontologies by using different visual representations, and hence, manipulating simultaneously these representations. Finally, we also show how this could be used to partially visualise arbitrary OWL 2 ontologies [Horrocks et al. 2006].

This work is structured as follows. Section 2 reviews some preliminary concepts related to the KF metamodel and the tool crowd. Section 3 presents and details the proposed framework together with the methodology to exploit it. Section 4 shows case study based on a realistic ontology. Finally, section 5 details related approaches, and section 6 concludes the work and presents future directions.

2. Preliminaries

In this section, we briefly describe the preliminary concepts of our work. We give descriptions of both the Keet-Fillottrani (KF) metamodel and crowd, which is a web tool for ontology development tasks.

2.1. KF Metamodel

The intention behind the ontology-driven KF metamodel [Keet and Fillottrani 2015] is to provide interoperability, integration and conversion of conceptual data models represented in different languages. For this purpose, the metamodel specifies an approach for transforming models in one language into other ones, and to be able to assert semantically proper links between them. It unifies all main static entities and constraints of UML Class Diagrams, EER, and ORM 2. The KF hierarchy is currently divided into `Entity Types`, `Relationships` (such as `Attributes` and `Subsumptions`), `Roles` and `Constraints`. In this way, the metamodel covers all the native feature of the conceptual data model languages (CDMLs), and it allows to represent models of different languages in a single unifying model. Its structure and complete formalisation in FOL can be found in [Fillottrani and Keet 2014b].

The KF metamodel defines four groups of interoperability rules [Fillottrani and Keet 2014a] divided into *1:1 mappings*, *transformation*, *approximations*, and *no alternatives*. *1:1 mappings* are those where the elements are the same from an ontological point of view and as a consequence, the conversions are in simple steps. One of the simplest cases is `Subsumption`. Secondly, *transformations* involve elements which are essentially the same but not from a syntax point of view. Such transformations take place, for instance, from ORM 2 value types to UML and EER attributes. *Approximations* are special kinds of rules where modellers are required to accept or reject the conversions or links. These rules are based on patterns which could lead to different outcomes. Finally, since these languages do not have the same expressiveness, some of their features cannot be either represented nor approximated in a target language. As an example, the conversion from ORM 2 compound cardinality to UML is not possible.

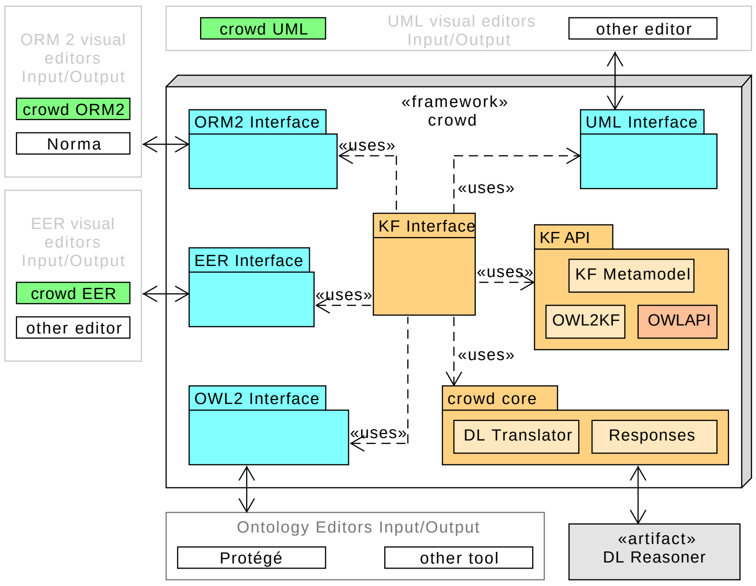

Figure 1. New framework crowd **for ontology engineering**

2.2. crowd Tool

crowd [Braun et al. 2020] is a visual web tool for ontology engineering tasks. The intention behinds crowd is to assist users to design OWL ontologies from standard CDMLs and visual support for developers. Its complete architecture has been described in [Braun et al. 2019b].

The leverage of automated reasoning is enabled by a precise semantic definition of all the elements of the diagrams. Hence, diagrams constraints are internally translated into a logic-based formalism capturing typical features of models. The tool is fully integrated with a powerful logic-based reasoning server acting as a background inference engine. Unlike other tools, crowd is based on a deduction-complete notion of reasoning support relative to the diagram graphical syntax. Thus, users will see the original model graphically completed with all the deductions and expressed in the graphical language itself. This includes checking class and relationship consistency, discovering implied class and cardinality constraints. First versions of crowd only focus on visual modelling of schemes, while it does not consider individuals yet.

Currently, crowd enables ontology development tasks by using UML diagrams and supporting automated reasoning over them. Furthermore, it presents Semantic Web Compatibility, the tool ensures compliance with W3C standards by allowing defining global naming schemes as well as exporting specifications OWL 2 to interoperate with other tools.

3. Putting together Diagrams, Metamodelling and Reasoning on the web

In this section, we describe the main components of our new framework. We first present its architecture, where both the KF metamodel and the new crowd core are integrated. Finally, we explain the methodology underlying to our approach and show the capabilities of the framework in a behavioural model. The suite of tools involved in this framework is composed of a core PHP, two visual tools for both UML and ORM 2 diagrams and two Java APIs. All of the tools are open source and are available at: `https://bitbucket.org/gilia/workspace/projects/CROW`.

3.1. Framework

Figure 1 depicts the module view style of the proposed architecture. The `A <<uses>> B` relationship defines the interaction between modules. Each module provides a set of responsibilities aiming to define the role of such a module in achieving certain functionality. This view details how these units ensemble to form one greater structure. The framework is structured as follows.

(i) The external visual editors for UML, EER and ORM 2. Each one interacts with the framework by sending the diagrams using the HTTP protocol. Currently, only the UML and ORM 2 editors provided by crowd have been integrated to the framework, however, other applications dealing with conceptual modelling languages for ontology development communicate with the core by giving their models to the respective interfaces.

(ii) The interfaces are the entry points to the framework. They process the inputs and outputs according to the schemes accepted by the `KF interface`. From the implementation point of view, these schemes are JSON objects with the primitives of the visual models. Aiming at using crowd to import OWL 2 ontologies, the framework includes an OWL 2 interface which differs from the other ones. In particular, it only loads an OWL document, which is after parsed and manipulated by the OWL API [Horridge and Bechhofer 2011]. Currently, the functionalities associated with the manipulation of OWL 2 ontologies are limited to extract atomic class, the whole hierarchy of classes and sub-hierarchies from a particular super class.

(iii) The `KF API` is based on the underlying theory of the KF metamodel. This component instantiates the metamodel to convert between different diagrams given in a particular CDML. The KF API implements the following 1:1 Mapping rules: *Relationship, Role, Object Type, Subsumption, Object Type cardinality, Completeness and Disjointness between Object Types, and Subset constraint* and *Mandatory*. Regarding transformation rules, *UML Attribute from/to ORM Value Type* is the only rule implemented. Lastly, the `KF API` also implements an initial version of an `OWL2toKF` component, in charge of generating KF instances from OWL 2 documents given as input and parsed by using the OWL API.

(iv) The last module is `crowd core`. Its main purpose is to manipulate and encode KF instances as DL Knowledge Base (KB). Moreover, it implements OWL 2 builders and the features to deal with off-the-shelf reasoners.

All of these modules are orchestrated by the `KF Interface`, which receives the external inputs and redirect them to the remaining modules according to the requirements from users.

3.2. Methodology

Our methodology exploits visualisation, metamodelling and reasoning for dealing with ontology development tasks. In particular, it enables custom visualisations but keeping only one common and unified model. Figure 2 illustrates an OCBC model with the flow

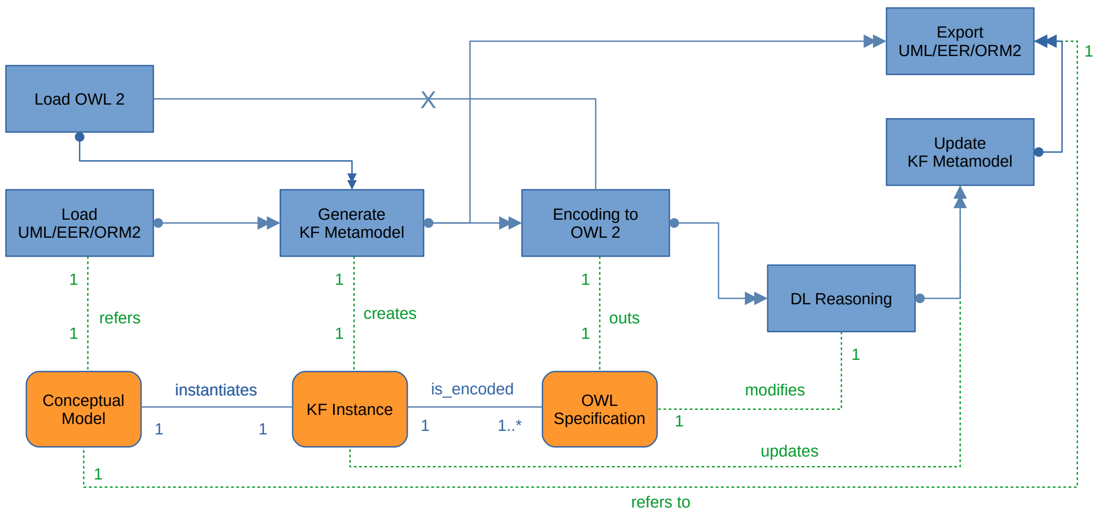

Figure 2. OCBC model [Artale et al. 2019] illustrating the flow of tasks in our framework. Double arrow between two tasks indicates that if the first task executes, the second one must be executed afterwards.

of tasks in our framework. Here we show the interaction between these tasks (in blue) and between the underlying data model (in orange). We consider the below scenarios.

(i) In the first scenario, users draw a scheme using any of the CDML supported, which is automatically mapped to the KF metamodel and thus generating the respective instance (applying the metamodel interoperability rules): `Load UML/EER/ORM2` → `Generate KF Metamodel`. According to the users' choice, this instance could be exported again to other CDML: `Generate KF Metamodel` → `Export UML/EER/ORM2`; or encoded into OWL 2 to be checked. In the latter case, the metamodel instance could or not be updated with the reasoning results before being exported and visualised in one of the CDML supported: `Generate KF Metamodel` → `Encoding to OWL 2` → `DL Reasoning` → `Update KF Metamodel` → `Export UML/EER/ORM2`.

(ii) In the second scenario, users load an OWL document, which is automatically mapped to the KF metamodel and thus generating the respective instance: `Load OWL 2` → `Generate KF Metamodel`. Here, the KF instances are generated by mapping OWL axioms to KF entities, previously parsed by the OWL API. Once the KF instance has been generated, the only possible task to be executed is to export it to one of the CDML supported: `Generate KF Metamodel` → `Export UML/EER/ORM2`. The crossed line between `Load OWL 2` and Encoding to OWL 2 indicates that if `Load OWL 2` is executed then `Encoding to OWL 2` is never executed.

As far as we are aware, there is not any exact match between arbitrary OWL 2 ontologies and the KF metamodel [Keet and Fillottrani 2013], i.e. mapping rules to convert OWL 2 primitive into KF ones. Therefore, we sketch an approximation based on the use of new rules, similar to 1:1 mapping ones, as starting point. To see the complexity of this mapping, we give an example with the case of OWL 2 `object properties` and KF `relationships`. Just to exemplify, we consider the **(R1)** Rule

(**R1**) Association $\xRightarrow{\text{UML to KF}}$ `Relationship`

in: Association(AssociationEnd: Class, AssociationEnd : Class) (1)

out: AssociationEnd → `Role` (2)

out: Association → `Relationship` (3)

out: Class → `Object Type` (4)

out: `Relationship(Role:Object Type, Role:Object Type)` (5)

Figure 3. (R1) Rule UML to KF from [Fillottrani and Keet 2014a].

from [Fillottrani and Keet 2014a] shown in Figure 3 (UML to KF), where we can see that for instantiating a KF relationship we should have at least two participating entities. Moreover, we need two roles.

So let us suppose that an ontology defines the `hasLocation` object property, whose domain and range are `BioEvent` and `Location`, respectively. Thus, the **(R1)** input is hasLocation(AssociationEnd: BioEvent, AssociationEnd: Location) (Figure 3, (1)). Following the rule structure, we should map the object property as a metamodel relationship `hasLocation` (Figure 3 (3)). As we said, the definition of a relationship requires of other associated primitives: `Role` and `Object Type`. Thus, both object property domain `BioEvent` and range `Location` can be mapped to `Object Type` (Figure 3 (4)), however, two *fresh* roles, `bioEventRole: Role` and `locationRole: Role` must be created to complete the definition of such relationship and instantiate (Figure 3 (2)). Therefore, adding those *fresh* roles implies that a new OWL 2 specification, different from the first one imported, could be generated.

4. Case Study

BiGe-Onto [Zárate et al. 2019] is an ontology-based system to manage information from Biodiversity and Biogeography domains, using standards such as the `Darwin Core` and `GeoSPARQL`. It is composed of a *BiGe-Onto* architecture; a conceptual model called *BiGe-Onto*; an OWL 2 operational version of *BiGe-Onto*; and an integrated dataset for its exploitation through a SPARQL endpoint.

Figure 4 depicts the current model of *BiGe-Onto*, which has been modelled in `crowd-UML`. There, we can identify each ontology reused to build it. The corresponding URIs of namespaces have been also loaded in the tool. Concepts extracted from `Darwin Core` vocabulary models events, where occurrences referencing organisms have been identified along with their respective locations and regions. On the same hand, the information related to these locations and regions are modelled by reusing concepts from `GeoSPARQL` and `ENVO`. Finally, events have date and time descriptions given by `Time` ontology, and datasets and persons by `VoID` and `FOAF`, respectively. This model has been used to guide the process of mapping the raw data to RDF. Further information about the case study, such as demos and images, can be found at `http://crowd.fi.uncoma.edu.ar/ontobras/`.

Converting from UML to ORM 2

Our first case study is about applying interoperability rules to convert an initial UML model of *BiGe-Onto* into an ORM 2 model. This scenario involves running the following tasks depicted in the Figure 2: `Load UML`, `Instantiate KF Metamodel`, and

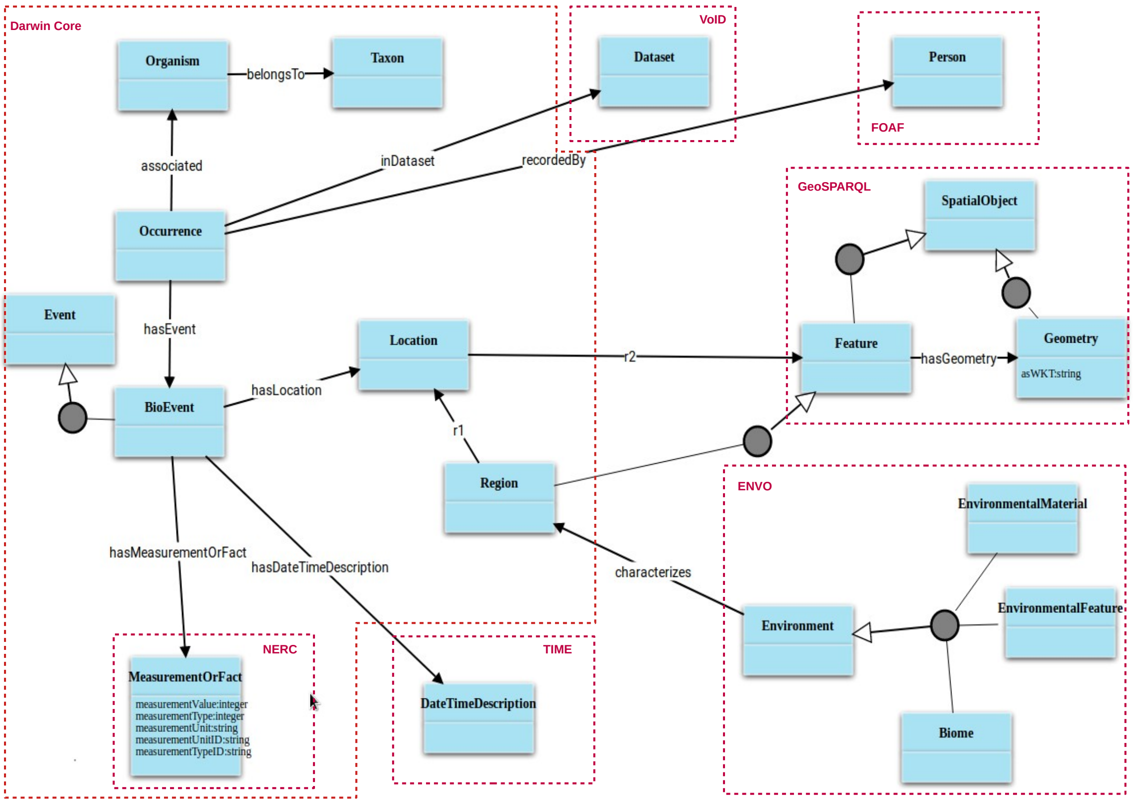

Figure 4. Visualisation of *BiGe-Onto* in `crowd-UML`

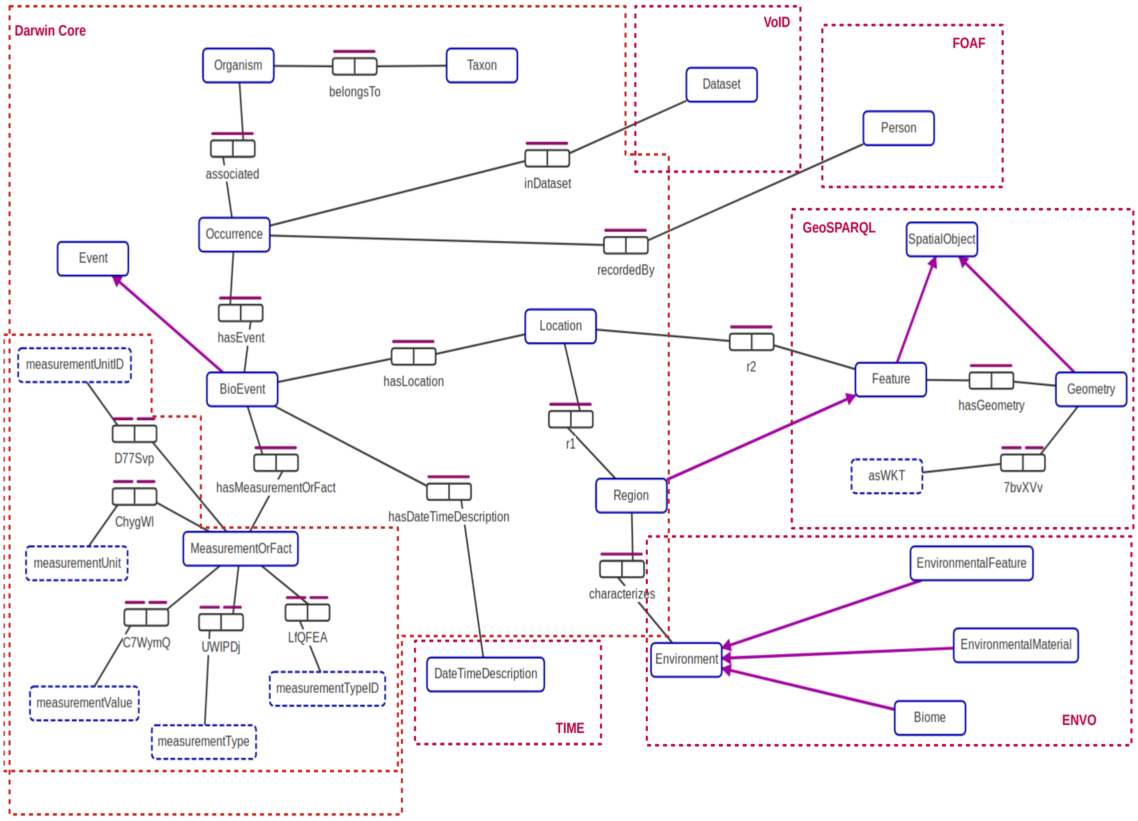

Figure 5. Visualisation of *BiGe-Onto* in `crowd-ORM 2`

`Export ORM 2`. Both the first and the last tasks are depicted in Figure 4 and Figure 5, respectively. The crucial task here is to instantiate the KF metamodel, which imply to apply its interoperability rules.

As can be seen from Figure 4, the model contains 18 classes, 7 generalisations, 11 associations (with their respective roles), and 6 attributes distributed between the classes *MeasurementOrFact* and *Geometry*. Using the *1:1 Mapping* rule we can generate the following entities: 18 `Object Type`, 11 `Relationship`, 22 `Role`, 7 `Subsumption`, and 22 `Object type cardinality`, making a total of 80 rules applied to generate our instance. To illustrate it, below we show the case of `Relationships(R)` rule. For the example, we use the classes *Environment*, *Region*, and the association *characterizes* from the *BiGe-Onto* model in Figure 4.

(R) UML to KF

$$
\begin{aligned}
characterizes : Association &\rightarrow \mathtt{characterizes} : \mathtt{Relationship} \\
environmentEnd &\rightarrow \mathtt{environmentRole} : \mathtt{Role} \\
regionEnd &\rightarrow \mathtt{regionRole} : \mathtt{Role} \\
characterizes &\rightarrow \mathtt{characterizes} : \mathtt{Relationship} \\
Environment &\rightarrow \mathtt{Environment} : \mathtt{ObjectType} \\
Region &\rightarrow \mathtt{Region} : \mathtt{ObjectType} \\
\\
environmentEndMult : Multiplicity &\rightarrow \mathtt{environmentEndMult} : \mathtt{OTCardinality}^{2} \\
0 &\rightarrow \mathtt{0} : \mathtt{MinOTCardinality} \\
N &\rightarrow \mathtt{N} : \mathtt{MaxOTCardinality} \\
regionEndMult : Multiplicity &\rightarrow \mathtt{regionEndMult} : \mathtt{OTCardinality}^{2} \\
0 &\rightarrow \mathtt{0} : \mathtt{MinOTCardinality} \\
N &\rightarrow \mathtt{N} : \mathtt{MaxOTCardinality}
\end{aligned}
$$

(R) KF to ORM 2

$$
\begin{aligned}
\mathtt{characterizes} : \mathtt{Relationship} &\rightarrow characterizes : FactType \\
\mathtt{environmentEnd} &\rightarrow environmentRole : Role \\
\mathtt{regionEnd} &\rightarrow regionRole : Role \\
\mathtt{characterizes} &\rightarrow characterizes : FactType \\
\mathtt{Environment} &\rightarrow Environment : ObjectType \\
\mathtt{Region} &\rightarrow Region : ObjectType \\
\\
\mathtt{environmentEndCard} : \mathtt{Cardinality} &\rightarrow environmentEndCard : OTCardinality^{2} \\
\mathtt{0} &\rightarrow 0 : MinOTCardinality \\
\mathtt{N} &\rightarrow N : MaxOTCardinality \\
\mathtt{regionEndCard} : \mathtt{Cardinality} &\rightarrow regionEndCard : OTCardinality^{2} \\
\mathtt{0} &\rightarrow 0 : MinOTCardinality \\
\mathtt{N} &\rightarrow N : MaxOTCardinality
\end{aligned}
$$

Once the KF metamodel has been instantiated, the next step in the conversion process is the application of interoperability rules to generate an ORM 2 model from the KF

[2] ObjectTypeCardinality

metamodel instance. The conversion from the metamodel to ORM 2, using *1:1 Mapping* rules, will generate an equal amount of entities as the metamodel instance but using ORM 2 primitive names.

Reasoning over Metamodel Instances

In addition, to provide customisable visualisations from the utilisation of a unified model, we want to add support for automated reasoning over such common instance. This scenario corresponds to the flow `Load UML` → `Generate KF Metamodel` → `Encoding to OWL 2` → `DL Reasoning` → `Update KF Metamodel` → `Export UML` defined for our methodology. In particular, it involves encoding each metamodel instance as an OWL specification, and thus to be sent to off-the-shelf DL reasoners for checking for satisfiability and discovering (possible) implicit assertions.

Considering again our initial *BiGe-Onto* model and the OWL formalisation of the whole metamodel[3], we consider the DL semantics of UML generalisations and an association of *BiGe-Onto* as shown below.

(S) KF to DL

$$\begin{aligned}
\texttt{Feature} : \texttt{ObjectType} &\rightarrow \mathit{Feature} : \mathit{Class}\\
\texttt{Geometry} : \texttt{ObjectType} &\rightarrow \mathit{Geometry} : \mathit{Class}\\
\texttt{SpatialObject} : \texttt{ObjectType} &\rightarrow \mathit{SpatialObject} : \mathit{Class}\\
(\texttt{Feature}, \texttt{SpatialObject}) : \texttt{Subsumption} &\rightarrow (\mathit{Feature}, \mathit{SpatialObject}) : \mathit{SubClassOf}\\
(\texttt{Geometry}, \texttt{SpatialObject}) : \texttt{Subsumption} &\rightarrow (\mathit{Geometry}, \mathit{SpatialObject}) : \mathit{SubClassOf}
\end{aligned}$$

DL

$$\begin{aligned}
\mathit{Feature} &\sqsubseteq \mathit{SpatialObject}\\
\mathit{Geometry} &\sqsubseteq \mathit{SpatialObject}
\end{aligned}$$

The first mapping (S) KF to DL takes the metamodel instance generated from UML composed generalisation between the `SpatialObject`, `Feature` and `Geometry`. Such instance generates two `Subsumption` primitives and the respective `Object Types`. In DL, we simply write two concept inclusion between these atomic concepts.

Exploring OWL 2 Ontologies

There exist many ways to express requirements and test ontologies to understand the content of them [Blomqvist et al. 2012, de Almeida Falbo 2014] and thus reusing them to extend other ontologies. However, not all entities or relationships are of interest to be reused, implying that we should be aware of the structure of such related ontologies. This last case describes the use of the metamodel as a common model to interoperate between OWL 2 and CDML-based visualisations, aiming to explore OWL 2 ontologies for identifying those entities of interest. The scenario corresponds to the following tasks from our OCBC model: `Load OWL 2` → `Generate KF Metamodel` → `Export ORM2`. Thus, we will generate an instance of the KF for the structure of interest of the loaded ontology and finally, a visualisation of such instance in UML, EER or ORM 2. As we already justified in section 3.2, we will cancel the task of encoding again to OWL 2.

We describe the case of extending *BiGe-Onto* by re-using entities from FaBiO[4] ontology [Peroni and Shotton 2012] for describing publications related to findings about

[3]`http://crowd.fi.uncoma.edu.ar/KFDoc/`

[4]`http://www.sparontologies.net/ontologies/fabio`

data of *BiGe-Onto*. FaBiO is an ontology for recording and publishing on the Semantic Web descriptions of entities that are published or potentially publishable. Then, let us suppose that we want to explore the kinds of publications, for instance lecture notes, chapters, etc. From the FaBiO documentation, we also know that these publications are expression of works published or potentially publishable so that the starting point is to look for the class `Expression` and its subclasses. For doing this task and after loading the respective OWL 2 specification of FaBiO, our framework generates a KF instance by executing these rules:

OWL 2
$$\begin{aligned}
fabio:Expression &\ \ a\ owl:Class\\
fabio:LectureNotes &\ \ a\ owl:Class\\
rdfs:subClassOf &\ \ fabio:Expression\\
fabio:Chapter &\ \ a\ owl:Class\\
rdfs:subClassOf &\ \ fabio:Expression
\end{aligned}$$

OWL 2 to KF
$$\begin{aligned}
fabio:Expression &\rightarrow \texttt{Expression}:\texttt{ObjectType}\\
fabio:LectureNotes &\rightarrow \texttt{LectureNotes}:\texttt{ObjectType}\\
LectureNotes.subClassOf:Expression &\rightarrow \texttt{playSub}:\texttt{Role}\\
fabio:Expression &\rightarrow \texttt{playSup}:\texttt{Role}\\
fabio:Chapter &\rightarrow \texttt{Chapter}:\texttt{ObjectType}\\
Chapter.subClassOf:Expression &\rightarrow \texttt{playSub}:\texttt{Role}\\
fabio:Expression &\rightarrow \texttt{playSup}:\texttt{Role}
\end{aligned}$$

As the OWL 2 to KF rule shows, OWL 2 classes are mapped to `Object Type`, while both `LectureNotes` and `Chapter` play the subclass role and `Expression` plays superclass role. Once a KF instance has been generated containing all the subsumptions, we could follow the flow to export this instance into an appropriate visual language.

5. Related Works

Several approaches have been proposed to address formalisation and implementation of suitable tools for ontology engineering tasks, that integrate visual multi-model, conversion among different CDMLs and automated reasoning. One of such works is presented in [Boyd and McBrien 2005], where authors developed a Hypergraph Data Model to relate models represented in ER, relational, UML, and ORM through transformation rules. However, the expressiveness of models is limited, since they omit roles, aggregation, weak entity types and several constraints. This approach can be considered as a platform for automating the process of transforming between different modelling languages, but currently, it is not implemented. Another work has been carried out by Halpin [Halpin 2004, Halpin 2002], where diverse mappings from ORM to UML and from ORM to ER are analysed. However, these last ones are only theoretical and are presented as separated approaches as well.

Going one step further, an Universal Conceptual Modelling (UCM) Framework is proposed in [Sportelli 2017] providing reasoning services and import/export of diagrams in different languages like ORM, UML and ER. Even this approach is similar to our framework, there exists some differences: the visualisation of EER, ORM and UML conceptual languages integrated into our framework, and the interaction between such visualisations and partial views OWL axioms. Furthermore, our implementation allows users to extend visual model expressive with OWL coding and manage namespaces, two important aspect in ontology development. Currently, the whole UCM framework is not implemented yet, but the author presents the application ORMiE [Sportelli 2016], which is a plugin for NORMA [Curland and Halpin 2010], to perform reasoning on ORM 2 conceptual scheme extended by derivation rules, augmenting ORM 2 standard expressibility.

GeRoMe [Kensche et al. 2007] is a Generic Role-based Metamodel which represents models from different modelling languages (such as EER, UML, XML Schema, OWL, SQL) in a generic way. Each model element in GeRoMe is labelled with a set of role objects that represent specific properties of the model element. This framework does not support ORM 2, it does not provide automated reasoning nor graphical functionalities. Atzeni et al. [Atzeni et al. 2011] present an automated model-independent approach for translating a model from one language to another, considering ER, UML, Object-Oriented and Object Relational languages. Translations are specified as Datalog rules and the source and target of the translation are exposed in a generic relational dictionary of constructs. However, a sequence of rules is required to translate from a representational language to another one (if possible). Visual features are basics.

Finally, two commertial visual tools have been considered: Astah [Astah Homepage 2020] and Visual Paradigm [Visual Paradigm Homepage 2020]. Astah supports UML and EER conceptual model languages, but does not support ORM. Astah Professional allows to convert ER diagrams to UML Class diagrams and back to ER diagrams again. Reasoning support is provided just for UML to check inconsistency between Class diagram models and Sequence diagram models. Visual Paradigm is tool suite for software development that provides UML, EER and BPMN visual modelling capabilities. Visual Paradigm supports generating UML class diagrams from ER diagrams. Both automated reasoning nor interoperability capabilities are provided either.

6. Conclusions and Future Works

We have shown the feasibility of using a unified model to implement visual independence in an ontology engineering tool by leveraging the theoretical KF metamodel and automated reasoning. To this end, we have provided a framework that integrates visual editing over EER, UML and ORM 2 conceptual modelling languages, along with a methodology to exploit it. Finally, we have demonstrated the applicability of our approach by introducing case studies on a realistic ontology.

The central aspect of the methodology is based on manipulating metamodel instances, which can be exported as visual models in diverse conceptual data modelling languages or mapped to OWL 2 to reason over them. Thus, separating the visual abstraction from the formal one. Through the case studies, we have analysed diverse scenarios where visualisation, metamodelling and reasoning are required. However, they also brought to light new challenges. In particular, those related to the interoperability with arbitrary OWL 2 ontologies and the scalability of the whole approach.

In conclusion, we foresee various future to improve the results obtained here. On one side, the obvious one is to continue the implementation of more interoperability rules. On the other hand, we plan to extend the *compatibility* of OWL 2 and the KF for dealing with arbitrary OWL 2 ontologies.

References

Artale, A., Calvanese, D., Kontchakov, R., Ryzhikov, V., and Zakharyaschev, M. (2007). Complexity of Reasoning in Entity Relationship Models. In *International Workshop on Description Logics*.

Artale, A., Kovtunova, A., Montali, M., and van der Aalst, W. M. P. (2019). Modeling and Reasoning over Declarative Data-Aware Processes with Object-Centric Behavioral Constraints. In *17th International Conference, BPM, Proceedings*, pages 139–156.

Astah Homepage (2020). `http://astah.net/features/convert-diagrams-and-models`.

Atzeni, P., Gianforme, G., and Cappellari, P. (2011). Data model descriptions and translation signatures in a multi-model framework. *Annals of Mathematics and Artificial Intelligence*, 63(3-4):287–315.

Berardi, D., Cali, A., Calvanese, D., and Giacomo, G. D. (2003). Reasoning on UML Class Diagrams. *Artifical Intelligence*.

Blomqvist, E., Sepour, A. S., and Presutti, V. (2012). Ontology Testing - Methodology and Tool. In *Knowledge Engineering and Knowledge Management - 18th International Conference, EKAW 2012. Proceedings*, pages 216–226.

Boyd, M. and McBrien, P. (2005). Comparing and Transforming Between Data Models Via an Intermediate Hypergraph Data Model. In Spaccapietra, S., editor, *Journal on Data Semantics IV*, pages 69–109, Berlin, Heidelberg. Springer Berlin Heidelberg.

Braun, G., Cecchi, L., and Fillottrani, P. (2019a). Taking Advantages of Automated Reasoning in Visual Ontology Engineering Environments. In *JOWO@BOG*.

Braun, G., Estevez, E., and Fillottrani, P. (2019b). A Reference Architecture for Ontology Engineering Web Environments. *Journal of Computer Science and Technology*, 19(01):e03.

Braun, G., Gimenez, C., Cecchi, L., and Fillottrani, P. (2020). `crowd`: A Visual Tool for Involving Stakeholders into Ontology Engineering Tasks. *KI - Künstliche Intelligenz*.

Cerans, K., Ovcinnikova, J., Liepins, R., and Sprogis, A. (2012). Advanced OWL 2.0 Ontology Visualization in OWLGrEd. In *DB&IS*, Frontiers in Artificial Intelligence and Applications. IOS Press.

Curland, M. and Halpin, T. A. (2010). The NORMA Software Tool for ORM 2. In *CAiSE Forum*, Lecture Notes in Business Information Processing. Springer.

de Almeida Falbo, R. (2014). SABiO: Systematic Approach for Building Ontologies. In *Proceedings of the 1st Joint Workshop ONTO.COM*.

Dudáš, M., Lohmann, S., Svátek, V., and Pavlov, D. (2018). Ontology visualization methods and tools: a survey of the state of the art. *The Knowledge Engineering Review*, 33.

Fillottrani, P., Franconi, E., and Tessaris, S. (2012). The ICOM 3.0 Intelligent Conceptual Modelling Tool and Methodology. *Semantic Web Journal*.

Fillottrani, P. R. and Keet, C. M. (2014a). Conceptual Model Interoperability: A Metamodel-driven Approach. In *Rules on the Web. From Theory to Applications - 8th International Symposium, RuleML@ECAI. Proceedings*.

Fillottrani, P. R. and Keet, C. M. (2014b). KF metamodel formalization. *CoRR*, abs/1412.6545.

Franconi, E., Mosca, A., and Solomakhin, D. (2012). ORM2: Formalisation and Encoding in OWL2. In *On the Move to Meaningful Internet Systems Workshops*.

Glimm, B. and Stuckenschmidt, H. (2016). 15 years of semantic web: An incomplete survey. *KI-Künstliche Intelligenz*, 30(2):117–130.

Halpin, T. A. (2002). Information Analysis in UML and ORM: A Comparison. In *Advanced Topics in Database Research, Vol. 1*.

Halpin, T. A. (2004). Comparing Metamodels for ER, ORM and UML Data Models. In *Advanced Topics in Database Research, Vol. 3*.

Horridge, M. and Bechhofer, S. (2011). The OWL API: A Java API for OWL Ontologies. *Semantic Web*.

Horrocks, I. (2011). Tool support for ontology engineering. In *Foundations for the Web of Information and Services*, pages 103–112. Springer.

Horrocks, I., Kutz, O., and Sattler, U. (2006). The even more irresistible SROIQ. In *In KR*, pages 57–67. AAAI Press.

Keet, C. M. and Fillottrani, P. R. (2013). Toward an Ontology-Driven Unifying Metamodel for UML Class Diagrams, EER, and ORM2. In *Conceptual Modeling - 32th International Conference. Proceedings*.

Keet, C. M. and Fillottrani, P. R. (2015). An ontology-driven unifying metamodel of UML Class Diagrams, EER, and ORM2. *Data Knowl. Eng.*, 98:30–53.

Kensche, D., Quix, C., Chatti, M. A., and Jarke, M. (2007). Gerome: A generic role based metamodel for model management. In *Journal on data semantics VIII*, pages 82–117. Springer.

Ong, D. and Jabbari, M. (2019). A review of problems and challenges of using multiple conceptual models. In *Proceedings of the 27th European Conference on Information Systems (ECIS)*.

Peroni, S. and Shotton, D. (2012). FaBiO and CiTO: Ontologies for describing bibliographic resources and citations. *Journal of Web Semantics*, 17:33 – 43.

Sportelli, F. (2016). NORMA: A Software for Intelligent Conceptual Modeling. In *JOWO- FOIS*.

Sportelli, F. (2017). Supporting Conceptual Modelling in ORM by Reasoning. In *European Conference on Advances in Databases and Information Systems*, pages 422–431. Springer.

Vigo, M., Bail, S., Jay, C., and Stevens, R. (2014). Overcoming the Pitfalls of Ontology Authoring: Strategies and Implications for Tool Design. *International Journal of Human Computer Studies*.

Visual Paradigm Homepage (2020). `https://www.visual-paradigm.com/`.

Zárate, M., Braun, G., Fillottrani, P. R., Delrieux, C., and Lewis, M. (2019). BiGe-Onto: An Ontology-Based System for Managing Biodiversity and Biogeography Data. *Applied Ontology*.