PhD Thesis proposal
Supporting Conceptual Modelling in ORM by Reasoning

Francesco Sportelli

Otto von Guericke Universität Magdeburg

Abstract Object-Role Modelling (ORM) is a framework for modelling and querying information at the conceptual level. It comes to support the design of large-scale industrial applications allowing the users to easily model the domain. The expressiveness of the ORM constraints may lead to implicit consequences that can go undetected by the designer in complex diagrams during the software development life cycle. To avoid these issues we perform reasoning on ORM diagrams in order to detect relevant formal properties, such as inconsistencies or redundancies, that cause a software quality degradation leading to an increment of development times and costs.
In order to activate the reasoning process over ORM diagrams, we show a relevant and decidable ORM fragment encoded into OWL. Furthermore, we extend the ORM formalisation by Derivation Rules, which are additional ORM constructs that capture some relevant information of the domain that cannot be expressed in standard ORM.
To this end, we illustrate the implementation of our formalisation in the tool UModel, which provides also an API system in order to be integrated into any conceptual modelling software; we provide a real-case study where the host software is NORMA.

Keywords: ORM, conceptual modelling, reasoning, rules

1 Overview

Conceptual modelling is a critical step during the development of a database system. It is the detailed description of the universe of discourse in a language that is understandable by users of the business domain. Object-Role modelling (ORM) is a conceptual language for modelling, which includes a graphical and textual language for specifying models, a textual language for formulating queries, as well as procedures for constructing ORM models, and mapping to other kinds of models like UML and ER. ORM is fact-oriented, i.e., it models the information in a way that it can be verbalized using sentences that are easily understandable by domain experts and even for those who are not familiar with IT in general. Unlike ER and UML, fact-oriented models are attribute-free, treating all facts (sentences) as relationships (unary, binary, ternary etc.) and this makes it more stable and adaptable to changing business requirements. For example, instead of

using the attributes Person.isSmoker and Person.hiredate, fact-oriented models use the fact types Person smokes and Person was hired on Date [1].
The ORM constraints expressiveness may lead to implicit consequences that can go undetected by the designer in complex diagrams; this may also lead to various forms of inconsistencies or redundancies in the diagram itself that give rise to the degradation of the quality of the design and/or increased development times and costs. The approach used to solve this issue involves the automated reasoning in order to detect inconsistencies and redundancies. Moreover, ORM diagrams can be equipped with Derivation Rules which are additional ORM constructs that are able to express knowledge that is not expressible with standard ORM. The usage of those rules brings to a further complexity; in order to perform the reasoning task even on those ORM diagrams equipped with those rules, we need to detect a decidable fragment.
We also introduce the state of the art starting from the first ORM formalisation until the most recent one. Then, we introduce an overview of the ORM language focusing on its main features like fact-oriented, the verbalisation and the graphical notation. The section concerning the research has four subsections:

1. ORM formalisation, we present a decidable fragment up to 11 ORM constraints;
2. ORM Derivation Rules formalisation, an extension of the previous fragment;
3. UModel, a tool designed to activate automated reasoning on any conceptual modelling software;
4. ORMiE, a tool which performs reasoning on ORM diagrams using UModel.

In the end, we present the list of what is still to be done in the frame of the PhD.

2 Related Work

The ORM formalisation started with Terry Halpin's PhD Thesis [2]. In the context of design conceptual and relational schemas, Halpin formalized the NIAM language that is the ancestor of ORM. In his thesis there is the first attempt to formalize a modelling language in order to perform the reasoning task, so the main objective is to provide formal basis for reasoning about conceptual schemas and for making decision choices. After the spreading of ORM and its implementation in NORMA [3], [4], ORM became more popular so the logicians' community took into account the possibility to formalize this very expressive language.

In 2007, Jarrar formalizes ORM using $\mathcal{DLR}_{ifd}$ [5], an extension of Description Logics introduced in [6]. The paper shows that a formalisation in OWL $\mathcal{SHOIN}$ would be less efficient than $\mathcal{DLR}_{ifd}$ because some ORM constraints cannot be translated (predicate uniqueness, external uniqueness, set-comparison constraints between single roles and between not contiguous roles, objectification n-ary relationships). In [7], Jarrar encodes ORM into OWL $\mathcal{SHOIN}$. Another formalisation of ORM in $\mathcal{DLR}_{ifd}$ was done by Keet in [8].

In 2009 OWL 2 was recommended by W3C Consortium as a standard of ontology representation on the Web bringing some benefits: it is the recommended ontology web language; it is used to publish and share ontologies on the Web semantically; it is used to construct a structure to share information standards for both human and machine consumption; automatic reasoning can be done against ontologies represented in OWL 2 to check consistency and coherency of these ontologies.

An ORM formalisation based on OWL2 is proposed by Franconi in [9], where he introduces a new linear syntax and FOL semantics for a generalization of ORM2, called ORM2plus, allowing the specification of join paths over an arbitrary number of relations. The paper also identifies a "core" fragment of ORM2, called ORM2zero, that can be translated in a sound and complete way into the ExpTime-complete Description Logic $\mathcal{ALCQI}$. [10] provides a provably correct encoding of a fragment of ORM2zero into a decidable fragment of OWL2 and it is discussed how to extend ORM2zero in a maximal way by retaining at the same time the nice computational properties of ORM2zero.

The most recent paper related to ORM formalisation is [11] where Artale introduces a new extension of $\mathcal{DLR}$, namely $\mathcal{DLR}^+$. This paper is strictly connected with this work because the logic $\mathcal{DLR}^+$ it is meant to represent n-ary relationships which are suitable for languages like ORM. The ORM implementation we use is an ongoing work based on $\mathcal{DLR}^+$. In particular, the decidable fragment we use is $\mathcal{DLR}^{\pm}$, obtained by imposing a simple syntactic condition on the appearance of projections and functional dependencies in $\mathcal{DLR}^+$. In the paper is also provided an OWL encoding and it is proved that $\mathcal{DLR}^{\pm}$ captures a significant fragment of ORM2.

Since this work is also focused on the formalisation of derivation rules, we need to mention OCL. OCL stands for Object Constraint Language, it is the declarative language for describing rules that apply to UML diagrams for defining constraints in order to support the conceptual modelling, like Derivation Rules for ORM. In [12] has been provided a formalisation of a fragment of this language and has been also proved the equivalence between relational algebra and the fragment with only FOL features, namely OCL_{FO}.

3 ORM

ORM stands for Object-Role Modelling. It is a language that allows users to model and query information at the conceptual level where the world is described in terms of *objects* (things) playing *roles* (parts in relationships) [13]. The idea behind ORM and its approach is that an object-role model avoids the need to write long documents in ambiguous natural language prose. It's easy for non-technical sponsors to validate an object-role model because ORM tools can generate easy-to-understand sentences. After an object-role model has been validated by non-technical domain experts, the model can be used to generate a class model or a fully normalised database schema. ORM main features are:

– *fact-oriented*, all facts and rules are modelled in terms of controlled natural language (FORML) sentences easy to understand even for non-technical users;
– *attribute-free*, unlike ER and UML, makes it more stable and adaptable to changing business requirements;
– *graphical*, it has a graphical notation implemented by the software NORMA;
– *formalised*, it has a clear syntax and semantics, so reasoning on ORM diagrams is enabled.

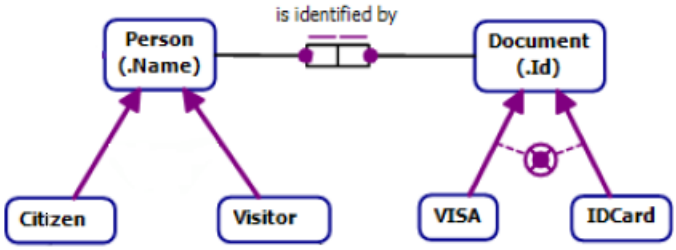

Figure 1: ORM diagram example

Unlike ER or UML, ORM makes no use of attributes in its base models; although this often leads to larger diagrams, an attribute-free approach has advantages for conceptual analysis, including simplicity, stability, and ease of validation. Attribute-free models with a controlled natural language facilitate model validation by verbalisation and population. Model validation should be a collaborative process between the modeller and the business domain expert who best understands the business domain. All facts, fact types, constraints and derivation rules may be verbalised naturally in unambiguous language that is easily understood by domain experts who might not be experts in the software systems ultimately used for the physical implementation.
The meaning of the diagram in Fig. 1 is the following: a person can be a citizen or a visitor; each person is identified by one and only one document which can only be either a visa or an id card. The entities are depicted by smooth rectangles and the relationships by a sequence of tiny boxes according to the

cardinality relationship. The purple dot represents a mandatory constraint, the dash on the tiny rectangle box is a uniqueness constraint, the equivalent of relational keys. The arrows among entities represents the ISA relationship. Finally, the circle with the cross inside means disjointness; the one with another circle inside means covering; the combination of these two is the circle we see between Visa and IDCard. The notation (.Name) and (.Id) inside Person and Document it is a graphical shortcut provided by NORMA for top level entities. Intuitively, it means that each person has a name and each document has an id. The corresponding FORML verbalization is the following:

```
Person is an entity type.
Citizen is an entity type.
Visitor is an entity type.
Document is an entity type.
VISA is an entity type.
IDCard is an entity type.
Person is identified by Document.
Each Citizen is an instance of Person.
Each Visitor is an instance of Person.
Each VISA is an instance of Document.
Each IDCard is an instance of Document.
Each Person has exactly one Document.
For each Document, at most one Person has that Document.
For each Document, exactly one of the following holds: that Document is
some VISA; that Document is some IDCard.
```

This feature turns out to be helpful during the modelling phase especially when the non-IT stakeholders interact with the software engineers in order to reach a mutual comprehension about the meaning of the diagram. For example, if the non-IT stakeholder detects unexpected sentences which do not reflect the software specifications, it is easy for the modeller to modify the interested part.

4 Motivation

Conceptual modelling tools are powerful systems which accelerate the development of the software, providing a set of powerful features to model a domain. They use conceptual modelling languages which are closer to the way we abstract the world in our cognition, making them ideal to model a domain. The limitation of the conceptual modelling tools is that they lack of semantics check capabilities. This limitation could lead to software degradation and unexpected software behaviours, especially for large-scale environment this could be a serious issue. The core idea of the dissertation is to provide a methodology in order to overcome this limitation. We believe that a conceptual modelling tool should not be limited to check the syntax of the diagrams, but the semantics too, because it could enhance the trust of the system, or, in the worst case could suggest the revision. The methodology involves the formalisation of conceptual

modelling languages and in the thesis the focus is on ORM. The ORM formalisation has been treated in years of research and still today it is a topic of interest. Formalising such language allows to activate reasoning procedures, carried out by Description Logics reasoners. The reasoning procedures are able to perform the semantic check over ORM diagrams, in this way is possible to manage the aforementioned limitation. The research follows two main tracks: the theoretical aspect, related to ORM and Derivation Rules formalisation; the other track is the implementation counterpart of the first track, since it concerns the creation of a set of tools in order to support the modeller.

4.1 Research goals

Outlined are the main goals of the research:

- **Goal 1**: providing an encoding for ORM in OWL;
- **Goal 2**: providing a formalisation for ORM Derivation Rules, both Subtype and FactType;
- **Goal 3**: implementation of a tool that adds reasoning capabilities to CASE tools;
- **Goal 4:** implementation of a real case study grounded on the entire methodology.

4.2 Novelty and relevance of research

Reaching the first goal, providing an OWL encoding for ORM, is the prerequisite to achieve the other goals since it constitutes the base of the methodology.

The second goal is an extension of the first one. Although several papers presented their own ORM formalisation, no one has taken into account the formalization of ORM derivation rules so far. Derivation rules are recent ORM constraints which are able to express knowledge that is beyond normal ORM capabilities, but this feature leads to an increase of expressiveness of the diagrams. For this reason, the challenge is to identify a decidable fragment in order to extend the reasoning even to those ORM diagrams equipped with those rules.

Reaching the third goal would provide to the community a tool which is able to overcome the aforementioned limitations of CASE tools. The direct impact of this goal involves various actors, such as the modellers, the database or software engineers. The versatility of the software makes possible to enrich any conceptual modelling software of reasoning capabilities, making the impact of the research in particular relevant for the industry world.

A successful result in the last goal could demonstrate the efficiency of the methodology applied to real cases.

5 Current State of Research

The main goal of the research aims to support the modeller during the design step of the software by reasoning, in order to detect constraints which can lead to

unexpected software behaviours, or to infer new knowledge, or more in general to increase the trust of the system. To apply the reasoning over an ORM diagram, we first need to formalise ORM. Therefore, we first provide the ORM decidable fragment that we detected so far, alongside its extension with ORM Derivation Rules. The contribution has also an implementation counterpart carried out by two tools: UModel(Unreal Model) and ORMiE(ORM Inference Engine). The former is a standalone tool which uses Fact++ as a background reasoning engine; the latter uses the UModel API system to add reasoning capabilities to NORMA (Natural ORM Architect) which is the official Microsoft Visual Studio plugin with full ORM implementation.

5.1 ORM encoding in OWL

ORM encoding in OWL is a process made by some steps that are summarized in the following lines and depicted in Fig.2.

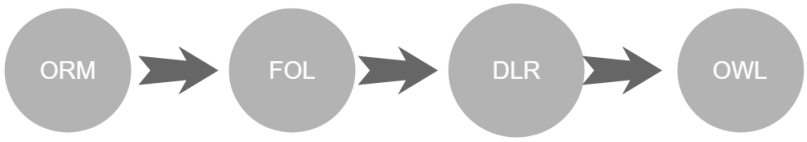

Figure 2: The formalisms involved in the process

At the first stage we take into account the syntax for every ORM constraint. Then, for each constraint, we assign a corresponding semantics in first order logic. At this point we rewrite the semantics using $\mathcal{DLR}^{\pm}$ because it is specifically designed to deal with n-ary relationships, in the limit of $\mathcal{DLR}^{\pm}$ expressibility; this means that some constraints are out of the fragment because they lead to undecidability. In the end, we apply the encoding from $\mathcal{DLR}^{\pm}$ to OWL. Once we have built the corresponding ontology, we can start the reasoning process. At the current time, the reasoning process supports the following inferences:

- hierarchy among predicates/objects
- equivalence among predicates/objects
- disjointness among predicates/objects
- simple mandatory constraint
- simple uniqueness constraint
- inconsistencies

Here are provided the involved formalisms and translations for the diagram in Fig.1, starting assigning the semantics to each constraint in first-order logic.

FactType
$\forall x, y.isIdentifiedBy(x, y) \rightarrow Person(x) \wedge Document(y)$

Mandatory
$\forall x.Person(x) \rightarrow \exists y.isIdentifiedBy(x,y) \wedge x = y$
Uniqueness
$\forall x,y.isIdentifiedBy(x,y) \rightarrow \exists^{\leq 1} z.isIdentifiedBy(x,z)$
$\forall x,y.isIdentifiedBy(x,y) \rightarrow \exists^{\leq 1} z.isIdentifiedBy(z,y)$
Subtype
$\forall x.Citizen(x) \rightarrow Person(x)$
$\forall x.Visitor(x) \rightarrow Person(x)$
$\forall x.Visa(x) \rightarrow Document(x)$
$\forall x.IDcard(x) \rightarrow Document(x)$
Exclusive
$\forall x.Visa(x) \rightarrow \neg IDCard(x)$
Exhaustive
$\forall x.Document(x) \rightarrow Visa(x) \vee IDCard(x)$

The next steps involves $\mathcal{DLR}^{\pm}$:
FactType
$isIdentifiedBy \sqsubseteq (\sigma_{1:Person} isIdentifiedBy) \sqcap (\sigma_{2:Document} isIdentifiedBy)$
Mandatory
$Person \sqsubseteq \exists[1] isIdentifiedBy$
Uniqueness
$\exists[1] isIdentifiedBy \sqsubseteq \exists^{=1}[1] isIdentifiedBy$
$\exists[2] isIdentifiedBy \sqsubseteq \exists^{=1}[2] isIdentifiedBy$
Subtype
$Citizen \sqsubseteq Person$
$Visitor \sqsubseteq Person$
$Visa \sqsubseteq Document$
$IDCard \sqsubseteq Document$
Exclusive
$Visa \sqsubseteq \neg IDCard$
Exhaustive
$Document \sqsubseteq Visa \sqcup IDCard$

Finally, it is possible to encode the ORM constraints in OWL.
FactType
$PRED\text{-}isIdentifiedBy \sqsubseteq PRED\text{-}isIdentifiedBy \sqcap \forall Q_1.Person \sqcap \forall Q_2.Document$
Mandatory
$PROJ\text{-}isIdentifiedBy\text{-}1 \equiv \exists Q_1^-.PRED\text{-}isIdentifieBy\text{-}1,2$
$SomeVisitor \sqsubseteq PROJ\text{-}isIdentifiedBy2\text{-}1$
Uniqueness
$UNIQ\text{-}isIdentifiedBy\text{-}1 \equiv \exists^{=1} Q_1^-.PRED\text{-}isIdentifieBy\text{-}1,2$
$PROJ\text{-}isIdentifiedBy\text{-}1 \equiv \exists^{=1} Q_1^-.PRED\text{-}isIdentifieBy\text{-}1,2$
$PROJ\text{-}isIdentifiedBy\text{-}1 \sqsubseteq UNIQ\text{-}isIdentifiedBy\text{-}1$
$UNIQ\text{-}isIdentifiedBy\text{-}2 \equiv \exists^{=1} Q_2^-.PRED\text{-}isIdentifieBy\text{-}1,2$
$PROJ\text{-}isIdentifiedBy\text{-}2 \equiv \exists^{=1} Q_2^-.PRED\text{-}isIdentifieBy\text{-}1,2$
$PROJ\text{-}isIdentifiedBy\text{-}2 \sqsubseteq UNIQ\text{-}isIdentifiedBy\text{-}2$

Subtype
$Citizen \sqsubseteq Person$
$Visitor \sqsubseteq Person$
$Visa \sqsubseteq Document$
$IDCard \sqsubseteq Document$
Exclusive
$Visa \sqsubseteq \neg IDCard$
Exhaustive
$Document \sqsubseteq Visa \sqcup IDCard$

All details regarding the formalisation are in an internal document that is planned to be published soon. 6.1.

5.2 Derivation rules

Derivation Rules are special ORM constructs which express knowledge that would otherwise not be expressible by standard ORM, so their expressibility reaches farer than standard ORM. Their purpose is to derive new information from other information, like triggers, stored procedures and views in SQL. The goal is to enable the reasoning on those rules in order to extend the reasoning even to those ORM diagrams equipped with Derivation Rules. There are two kind of Derivation Rules: the Subtype Derivation Rules and the Fact Type Derivation Rules. A Subtype Derivation Rule defines all the instances which belongs to a subentity by a set of constraints defined in the rule definition. The reason because those rules are applied on subentities is because in some diagrams the is-a relationship between entities is too weak to capture the entire desired semantics of the diagram; a FactType Derivation Rule is placed on the predicates, namely the ORM roles.
The Derivation Rules we are focusing on are Subtype Derivation Rules. We want to formalise those rules in order to activate the reasoning on those diagrams with Subtype Derivation Rules. To achieve this goal it is important to take into account that the reasoning lays on logic, so we need to find a way to encode derivation rules into a logical language. First of all we need to understand how a derivation rule is made from a structural point of view, in other words we need to detect a clear syntax. Then, at the syntax is assigned a corresponding semantics and in the end an encoding into a logical language is performed in order to made the reasoning possible. In [14] is provided the full methodology used to formalise those rules and their mapping into OWL.
We provide an example with the graphical notation implemented by NORMA. For example, the diagram in Fig. 1 does not tell us which are *exactly* the people in the entity Citizen and *exactly* the people in the entity Visitor. We only know that a person can be a citizen or a visitor, but in the ORM standard notation there are no constraints able to capture these sets. If we want to use this knowledge we have to use the ORM Derivation Rules like in Fig. 3.
As we can see, a derivation rule is defined by an asterisk on entities and a text which defines the meaning of the rule. We state that all the people which are

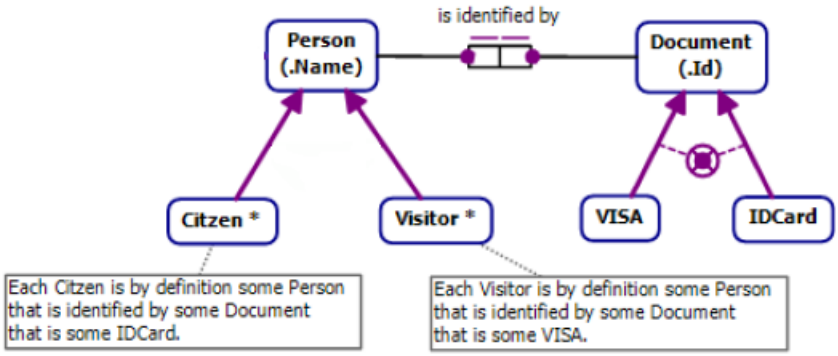

Figure 3: ORM diagram example with derivation rules

identified by an id card are citizens; all the people which are identified by a visa are visitors. It is important to observe that the text is not just a collections of words, instead it is in controlled-natural language format that is to say it is well defined by a precise syntax.

What can be the outcome of the diagram in Fig. 3? The answer is in Fig.4 We obtained a disjunction and covering between the entities Citizen and Visitor. The disjunction is inferred because there is no chance to find a common element between the entity Visa and IDCard. Since the derivation rules capture separately the two sets, visitors and citizens, even the corresponding entities have no element in common. What about the covering? Since Visa and IDCard cover Document and since Person has the mandatory constraint on the relationship *is identified by*, each person must participate to this relation; in addition to this, the two derivation rules ensure that the sum of the instances in Citizen and Visitor are exactly those who are in Person. So it is not possible to find an instance which is not in Citizen of in Visitor. To prove this, now we add the entity Illegal: people without documents, neither a visa nor id card. The outcome of the reasoning is shown in Fig.5. Illegal is red because it is an empty set. This means that in all consistent worlds the entity type Illegal is always empty, because there are no instances in Illegal which satisfy the rules of the diagram. Again, this is because Person has the mandatory constraint on the *is identified by* relationship and because the set of Person is already taken by the entities Citizen and Visitor. Therefore, there is no way an instance in Illegal could be in

Person. The counter-example is trivial: if we remove the mandatory constraint on Person then Illegal would not be inconsistent anymore.

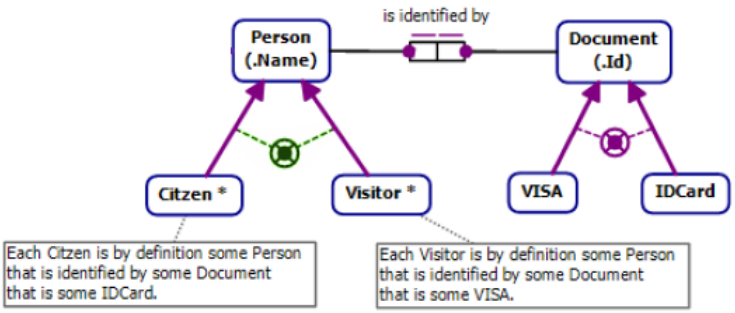

Figure 4: Inferred disjoint and covering constraints

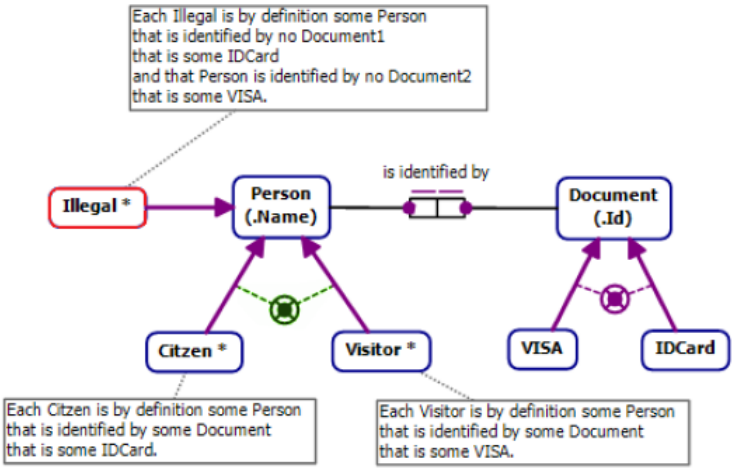

Figure 5: Illegal is inconsistent

5.3 UModel

The ambitious intention behind UModel is to build a framework that is able to add reasoning capabilities to any conceptual modelling software, supporting the most common languages such as UML, ER and ORM. In the thesis UModel is used for the ORM language and to enrich NORMA by reasoning capabilities. UModel is written in standard Java 8.0, so it can be used on Linux, Mac and Windows machines; a C# porting is also provided for .NET applications. It can be used as a standalone application or in a more interesting way by its API system, so in this way it can be easily integrated in other applications. UModel has several features: it provides API for developers, reasoning services, the import/export of ontologies and diagrams in different languages like ORM, UML and ER.

In Fig. 6 is shown the architecture of UModel. The idea behind this architecture is to isolate the core of the system being transparent to the user, which has to rely only on the API layer. The same applies for the host software that uses UModel, in our case NORMA/ORMiE. UController act as a wrapper which provides the necessary methods that are exposed to the user to interact with UModel. In order to create a diagram, the user starts calling multiple time the function *Tell*, that asserts the constraints in a specific object oriented data structure called *UModel*. After this step the module *UReasoner* is started. At the first stage it reads the model. Then, it constructs the OWL encoding based on the $\mathcal{DLR}^{\pm}$ formalisation. UReasoner uses the reasoner engine Fact++ to derive the fresh constraints and store them in the data structure *UDModel* (UDerivedModel). At this point, the user calls the *Ask* function to get the inferences encoded in object-oriented data structures. In this way, it is possible to use the information encapsulated in these objects to easily expose the inferences to the host application by few lines of code.

5.4 ORMiE

ORMiE (ORM Inference Engine) is an extension of NORMA, which is the official ORM-based Microsoft Visual Studio conceptual modelling tool [3]. ORMiE uses UModel API system, so it is just one example how UModel works on a target ORM-based software. ORMiE activates automated reasoning over ORM diagrams providing an interface where mistakes, redundancies or more in general new inferred knowledge are shown. A good feature of ORMiE is that is built on top of NORMA and Visual Studio, so it takes advantage of all nice features from Visual Studio being such a powerful tool for those who need to model a domain following the ORM methodology. Moreover, ORMiE is also able to perform the reasoning on ORM diagrams equipped with Derivation Rules.

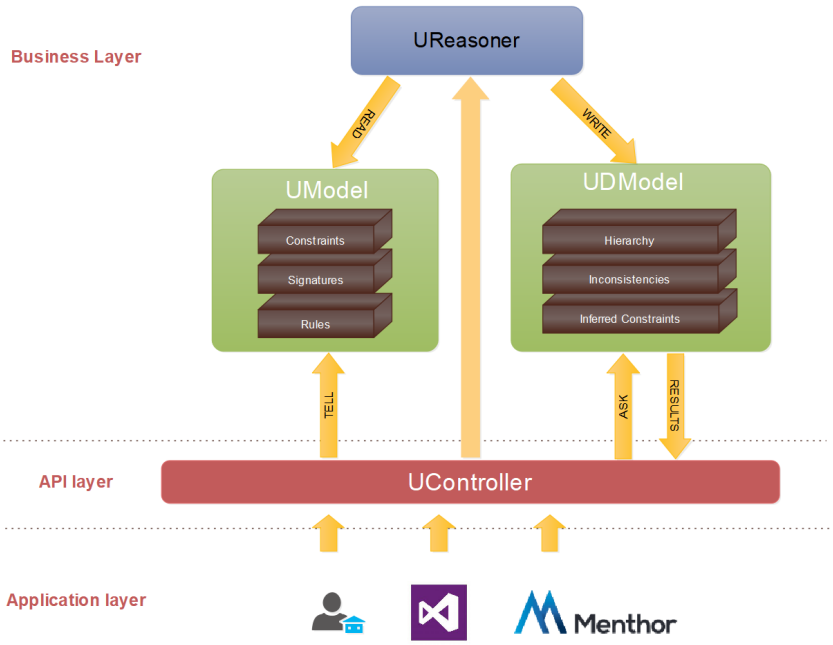

Figure 6: UModel Sequence Diagram

6 Ongoing Works

This section summarizes the tasks that are still to be done in the frame of my research.

6.1 ORM to OWL encoding

The formalisation involves a set of ORM constraints in order to apply the reasoning for a relevant and decidable fragment. In order to do so, the fragment is restricted to all that constraints that are expressible in $\mathcal{DLR}^{\pm}$ (everything except ring constraints), moreover, the set of constraints in the ORM diagram to be faithfully encoded in OWL by $\mathcal{DLR}^{\pm}$ must not violate the syntactic restriction over the projection signature graph which makes a $\mathcal{DLR}^{\pm}$ knowledge base decidable [11], [15]. The restriction says that the projections must not share common attributes. In Fig.7 is provided an example where the two uniqueness constraints generate two projections sharing one role.
At the current stage of the work, some constraints are yet not derived because they are hard to compute, therefore the reasoning process is slowed down. Solving this task is quite challenging because every constraints requires a specific approach in order to build an optimized algorithm. The current version of the OWL encoding is organised in an internal document which is still a draft. It

contains all the details concerning the formalisms involved for the encoding and the translations. The document will be published as soon as the remaining constraints will be finished.

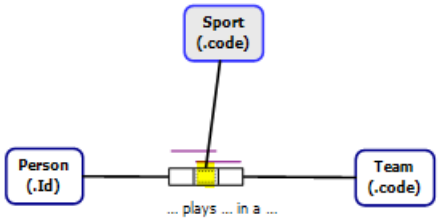

Figure 7: Two uniqueness constraints sharing one role

6.2 ORMiE release

The latest version of ORMiE has been tested on real-case ORM diagrams and it is quite stable. Now, ORMiE is able to perform the reasoning over ORM diagrams in the limit of the aforementioned fragment and the ORM Subtype Derivation Rules. What still is missing is the capability to deal also with ORM Fact Type Derivation rules, which requires from the implementation perspective the understanding of the complex structures inside NORMA. The difficulty is enhanced by the fact that NORMA completely lacks of documentation for the developers, forcing to perform a reverse engineering to understand how the data structures are made of.

6.3 UModel release

The current version of UModel is able to perform the reasoning over the ORM fragment explored so far plus ORM Derivation Rules. A deep work about optimization has already been implemented. We plan to improve UModel by adding a user interface that requires a specific study an the user experience with respect to the functionality of the reasoning services. In this way, the user can customize and choose which inferences are needed and which services to enable, giving more control over the whole system. The user interface is part of the API system, thus, is possible to attach it in every software hosting UModel. We plan to test UModel also on Menthor [16]. UModel will be downloadable soon with its documentation for developers.

6.4 Formalisation of the complete ORM Derivation Rules fragment

Both Subtype and Factype ORM Derivation Rules cover a decidable fragment. Until now we treated these fragments because of reasoning, but for the sake of completeness, we plan to formalise the full language.

7 Time Plan

Here is the time plan of the research according to the aforementioned ongoing works.

- **July 2018 - September 2018:** ORM to OWL encoding internal document release
- **July 2018 - September 2018:** Formalisation of the complete ORM Derivation Rules fragment
- **September 2018 - December 2018:** UModel release
- **September 2018 - December 2018:** ORMiE release
- **January 2019 - April 2019:** Dissertation writing

References

1. T. A. Halpin, "Object-role modeling: Principles and benefits," *IJISMD*, vol. 1, no. 1, pp. 33–57, 2010.
2. T. Halpin, *A Logical Analysis of Information Systems: static aspects of the data-oriented perspective*. PhD thesis, jul 1989.
3. M. Curland and T. A. Halpin, "The NORMA software tool for ORM 2," in *Information Systems Evolution - CAiSE Forum 2010*, pp. 190–204, 2010.
4. F. Sportelli, "NORMA: A software for intelligent conceptual modeling," in *Proceedings of the Joint Ontology Workshops 2016 Episode 2: The French Summer of Ontology co-located with the 9th International Conference on Formal Ontology in Information Systems (FOIS 2016), Annecy, France, July 6-9, 2016.*, 2016.
5. M. Jarrar, "Towards automated reasoning on ORM schemes," in *ER 2007, 26th International Conference on Conceptual Modeling*, pp. 181–197, 2007.
6. D. Calvanese, G. De Giacomo, and M. Lenzerini, "Identification constraints and functional dependencies in description logics," in *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4-10, 2001*, pp. 155–160, 2001.
7. M. Jarrar, "Mapping ORM into the SHOIN/OWL description logic," in *On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops, OTM Confederated International Workshops and Posters, AWeSOMe, CAMS, OTM Academy Doctoral Consortium, MONET, OnToContent, ORM, PerSys, PPN, RDDS, SSWS, and SWWS 2007, Vilamoura, Portugal, November 25-30, 2007, Proceedings, Part I*, pp. 729–741, 2007.
8. C. M. Keet, "Mapping the object-role modeling language ORM2 into description logic language dlrifd," *CoRR*, vol. abs/cs/0702089, 2007.
9. E. Franconi, A. Mosca, and D. Solomakhin, "The formalization of ORM2 and its encoding in OWL2," in *International Workshop on Fact-Oriented Modeling (ORM 2012)*, 2012.
10. E. Franconi and A. Mosca, "Towards a core ORM2 language," in *OTM Workshops*, vol. 8186 of *Lecture Notes in Computer Science*, pp. 448–456, Springer, 2013.
11. A. Artale and E. Franconi, "Extending DLR with labelled tuples, projections, functional dependencies and objectification," in *Proceedings of the 29th International Workshop on Description Logics*, 2016.
12. E. Franconi, A. Mosca, X. Oriol, G. Rull, and E. Teniente, "Logic foundations of the OCL modelling language," in *Logics in Artificial Intelligence - 14th European Conference, JELIA 2014* (E. Fermé and J. Leite, eds.), Springer, 2014.
13. T. A. Halpin and T. Morgan, *Information modeling and relational databases (2. ed.)*. Morgan Kaufmann, 2008.
14. F. Sportelli and E. Franconi, "Formalisation of ORM derivation rules and their mapping into OWL," in *On the Move to Meaningful Internet Systems: OTM 2016 Conferences - Confederated International Conferences: CoopIS, C&TC, and ODBASE 2016, Rhodes, Greece, October 24-28, 2016, Proceedings*, pp. 827–843, 2016.
15. A. K. Chandra and M. Y. Vardi, "The implication problem for functional and inclusion dependencies is undecidable.," *SIAM Journal on Compututing*, vol. 14, no. 3, pp. 671–677, 1985.
16. J. L. R. Moreira, T. P. Sales, J. Guerson, B. F. B. Braga, F. Brasileiro, and V. Sobral, "Menthor editor: An ontology-driven conceptual modeling platform," in *Proceedings of the Joint Ontology Workshops 2016 Episode 2: The French Summer*

of Ontology co-located with the 9th International Conference on Formal Ontology in Information Systems (FOIS 2016), Annecy, France, July 6-9, 2016., 2016.