Unifying Multilevel Modelling through Ontologies

Edelweis Rohrer[1], Paula Severi[2], and Regina Motz[1]

[1] Instituto de Computación, Facultad de Ingeniería, UdelaR, Uruguay
[2] Department of Computer Science, University of Leicester, England

Abstract. In the last decades, the multilevel problem has received increasing attention in the conceptual modelling and semantic web communities. Recently, we proposed a solution to this problem in the context of ontological modelling which consists in extending the Web Ontology Language OWL with a new multilevel constructor that equates instances to classes. In this work we highlight the advantages of exploiting the reasoning capabilities of OWL ontologies with the proposed multilevel constructor by analizing requirements from a real-world application on the accounting domain.

1 Introduction

Multilevel modelling is the conceptual modelling problem of having classes that could be instances of other classes (called metaclasses) or form part of metaproperties (properties between metaclasses). This is a relevant problem for many areas such as model-driven software development and ontology design [15,8,4]. Several works promote the use of ontologies for improving the theory and practice of conceptual modelling [7,15]. In general, we can group the *ontology-based conceptual modelling* works in four major categories: (i) works that propose the use of ontologies to develop modelling languages for design patterns and simulators [8,9], (ii) works that define methodologies to transform relational databases to ontologies [10,1,14], (iii) those that apply some formal theory [3,4,11] and (iv) those that extend the syntax and the reasoners of the web ontology language OWL to automatically check modelling consistency, e.g. [13,11]. In this last direction, we have already proposed a solution to the multilevel modelling problem which consists in extending OWL with a new constructor that equates instances to classes [13].

In this work, we show the usefulness of our multilevel constructor through a real-world application on the accounting domain. This is an interesting application because it shows the necessity of modelling (at least) two levels of knowledge: one for accounting domain experts (who are in charge of defining the system requirements) and another for operators (who are in charge to apply the daily actions according to the defined requirements). In fact, on one hand, experts visualize the whole organization landscape and are interested in establishing the business rules for each work procedure. For them, a procedure is naturally represented as an *instance* (an atomic object). On the other hand, a procedure for an operator is better represented as a *set of instances* (a complex object), since they execute the same procedure many times.

The main contributions of this work are twofold. First, a key point of our multilevel modelling approach is that by extending the ontology language and the reasoner, it is possible to automatically validate the consistency of a multilevel ontology as well as to automatically infer such level. Second, we unify the two different knowledge levels (experts and operators) by expressing common patterns of business rules using meta-properties (properties on classes). For this, we also introduce another constructor MetaRule2 that allows the automatic addition of several constraints following a common pattern. The results of the present work can be applied to other scenarios, in particular to those in which the flexible management of business rules is an important requirement.

Related Work. [2] formalizes the concept of multilevel connections and coins the term *clabject* to emphasize that there are classes that are also objects. Similarly, [4] combines the UFO ontology with a first-order logic, called MTL, to model multilevel connections. These proposals require that each class or individual explicitly specify its level of abstraction, for instance, 0 for atomic objects, 1 for classes, 2 for meta-classes, and so on. On the contrary, by infering the level our solution does not put any burden on the ontology designer who may be rather concerned with the business to be modelled than with the ordinal number of each abstraction level. Related work on description logics (the logical foundations of OWL) extended with metamodelling capabilities has an extense literature, due to space restrictions, we refer to [13] for a thorough comparison.

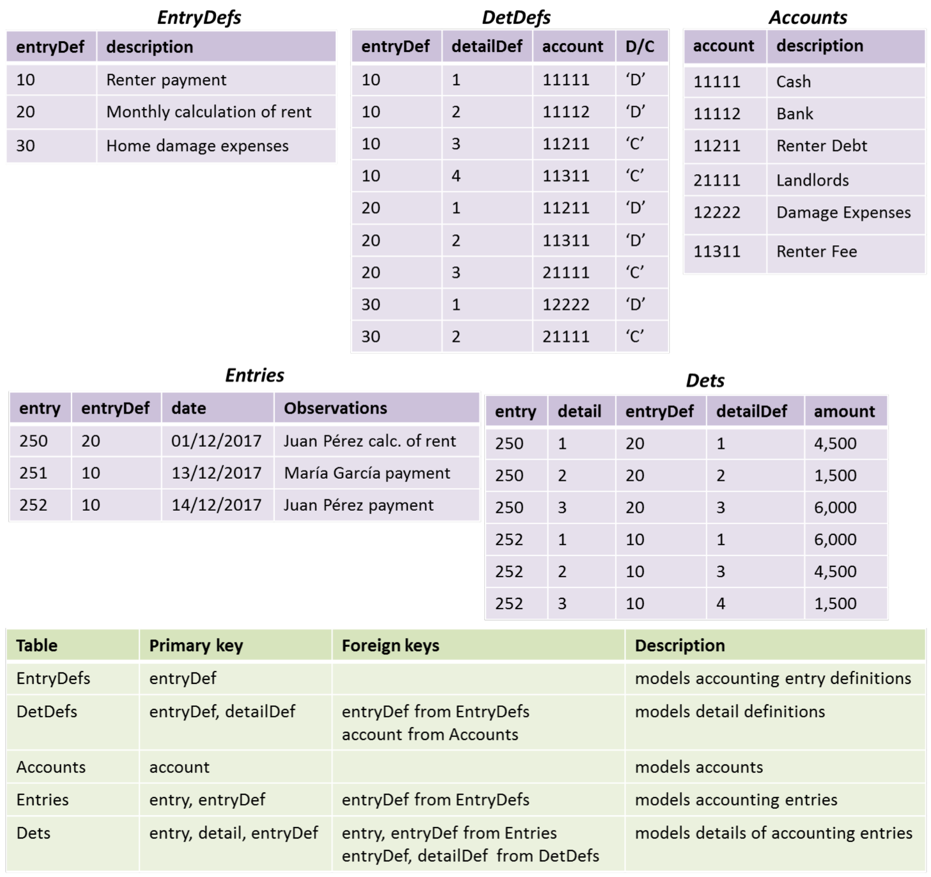
Organization. The remainder of this paper is organized as follows. Section 2 introduces the real-world case study and its main requirements. Section 3 describes two design alternatives for the case study using ontologies, and analyzes the degree of accomplishment of the set of requirements. Finally, Section 4 presents some conclusions and future work.

2 A real-world case study of an accounting information system

This section explains the main requirements behind the accounting module of an information system called " Integrated Rental Guarantee Management System" (SIGGA) at the General Accounting Agency of the Ministry of Economy and Finance in Uruguay[3] where the first author works. Uruguayan government acts as a guarantor for employees who want to rent a property. The underlying business of SIGGA consists in home rental contracts that are signed between landlords and renters who are employees. The application helps manage renter payments, as salary discounts or direct cash payment, and the corresponding payments to landlords. The accounting module is in charge of recording the accounting entries for the business rules of SIGGA, following the ALE-based classic double entry bookkeeping to represent incremental and decremental events of assets, libialities and equity as credits and debits [12]. The ALE-based system has several requirements such as accounting entries that record business transactions have at least one debit and one credit detail.

[3] Sistema Integrado de Gestión de Garantía de Alquileres, Contaduría General de la Nación, www.cgn.gub.uy

The application domain of SIGGA is modelled by a relational scheme and implemented in a relational database. Figure 1 depicts a simplified version of the relational tables of the application and their structural restrictions.



EntryDefs

entryDef	description
10	Renter payment
20	Monthly calculation of rent
30	Home damage expenses

DetDefs

entryDef	detailDef	account	D/C
10	1	11111	'D'
10	2	11112	'D'
10	3	11211	'C'
10	4	11311	'C'
20	1	11211	'D'
20	2	11311	'D'
20	3	21111	'C'
30	1	12222	'D'
30	2	21111	'C'

Accounts

account	description
11111	Cash
11112	Bank
11211	Renter Debt
21111	Landlords
12222	Damage Expenses
11311	Renter Fee

Entries

entry	entryDef	date	Observations
250	20	01/12/2017	Juan Pérez calc. of rent
251	10	13/12/2017	María García payment
252	10	14/12/2017	Juan Pérez payment

Dets

entry	detail	entryDef	detailDef	amount
250	1	20	1	4,500
250	2	20	2	1,500
250	3	20	3	6,000
252	1	10	1	6,000
252	2	10	3	4,500
252	3	10	4	1,500

Table	Primary key	Foreign keys	Description
EntryDefs	entryDef		models accounting entry definitions
DetDefs	entryDef, detailDef	entryDef from EntryDefs account from Accounts	models detail definitions
Accounts	account		models accounts
Entries	entry, entryDef	entryDef from EntryDefs	models accounting entries
Dets	entry, detail, entryDef	entry, entryDef from Entries entryDef, detailDef from DetDefs	models details of accounting entries

Fig. 1: Relational Tables for SIGGA accounting example.

From a conceptual point of view, there are two levels of the business: a *definitional level* and an *operational level* (general knowledge and concrete knowledge respectively, following the terminology in [15]). In the definitional level of the accounting module of SIGGA we identify different kind of accounting entries, called *entry definitions*, as depicted in table EntryDefs which are specified according to the business rules by a set of valid type of details, called *detail definitions*, registered in table DetDefs over accounts (table Accounts). Whereas, in the operational level, tables Entries and Dets register particular *accounting entries* and *details* respectively. For the sake of clarity we analize the accounting module of the SIGGA application with only three types of entries: "Monthly calculation of rent", Renter payment" and "Home damage expenses". For instance, the calculation of the monthly rent is performed each month for the renter

Juan Perez. On 01/12/2017 the SIGGA application registered the entry 250 in table Entries and the three first rows (details) of table Dets showing that Juan Perez must pay $4,500 of rent and $1,500 of renter fee (debited to the "Renter Debt" and "Renter fee" accounts), and that $6,000 must be acredited to the landlord account following the foreign keys (FK) restrictions from table DetDefs. Since Juan Perez paid the rent by cash on 14/12/2017, SIGGA registered the entry 252 and a set of records in table Dets with entryDef = 10 and detailDef 1, 3 and 4 that correspond to accounts "Cash", "Renter Debt" and "Renter Fee", according to FK resrictions from table DetDefs.

Nowadays, there are a set of basic requirements of the accounting module of SIGGA that are essential for its proper operation. The relational schema described in Figure 1 guarantees some of them, whereas others cannot be expressed by structural constraints. Moreover, there is also a set of desirable requirements that would add value to the system if they were conceptually modelled because they would contribute to a more explicit domain conceptualization and facilitate expert definition activities. Below we describe the list of requirements.

Req. 1. Each accounting entry has at least one debit detail and one credit detail. Given the structure of tables of the SIGGA application, this requirement is satisfied by non-structural restrictions, so it is satisfied only transactionally.

Req. 2. Each detail is either a debit or a credit (but not both). This requirement uses FK of Dets from DetDefs to ensure that the accounting detail has the corresponding attribute debit or credit.

Req. 3. Each detail has associated a single account. This requirement is directly satisfied by the FK of the table DetDefs from Accounts.

Req. 4. Each detail can be associated to a unique entry. This requirement is directly satisfied by the FK of the table Dets from Entries.

Req. 5. Each detail definition can be associated to a unique entry definition. This requirement ensures that accounting entry definitions are mutually independent. It is the reason why the SIGGA application does not link entry definitions to accounts directly. Using the concept of detail definitions the model is more flexible when definitions change. For instance, if the debit account changes for a given accounting entry definition, it is possible to keep old detail definitions marked as not current and introduce new detail definitions (to keep the history of definitions). This requirement is directly satisfied by the FK of the table DetDefs from EntryDefs.

Req. 6. Accounting entries have details for accounts in accordance with the definitional level. For instance, each time a renter pays a debt in cash the accounting entry must have details for accounts "Cash", "Renter Debt" or "Renter Fee" in accordance with the "Renter payment" entry definition. It is directly satisfied by the FKs of the table Dets (from Entries that has a FK from EntryDefs and from DetDefs that also has a FK from EntryDefs).

Req. 7. Provide mechanisms to classify accounting entries by relevant criteria. Common classification criteria are to classify all entries of a given kind (as renter payments) or with a given account (as “Cash”) at debit or credit. In the schema of Figure 1 these classifications are not explicitly modeled, they are implemented by SQL views.

Req. 8. Provide mechanisms to classify accounting entry definitions according to relevant criteria. To illustrate this requirement, we consider money availability accounts, as cash or bank. It is important to ensure that all deposits (debit details) and extractions (credit details) to/from them are recorded, to visualize the correct availability. So, the idea is to validate the completeness of definitions for a given account. A useful solution is to classify all accounting entry definitions that have cash or bank at debit or credit, to verify if all possible movements are defined. As for the previous requirement, it is possible to define SQL views.

Req. 9. Express relations between definitions to check their correctness. For instance, if the renter “Juan Pérez” incurres in a debt for damages in the house, it must be registered by the entry definition 30 (Figure 1) with a debit on the “Damage Expenses” account and a credit for the “Landlords” accounts. But there is no row in the table DetDefs associating the entry definition 10 (Renter payment) to the account “Damage Expenses” so the “renter payment” entry (to register that “Juan Pérez” payed the damage expenses) cannot be registered. What happens is that the expert did not include the “Damage Expenses” account at credit in the entry definition 10. So, we should validate that the “renter payment” entry definition to have credit details for all accounts that are debits in some entry definition that generates a renter debt, in the particular case, for are all asset accounts that are not of avalability. This is a validation that involves set of tuples from tables, it can be verified as a non-structural constraint (in the application code). It would be desirable that the relational schema had the capability of expressing such restrictions.

Req. 10. Minimize the impact of changing accounting entry definitions. Suppose that at a certain time, experts decide that for the definition of the “Renter payment” entry instead of the account “Renter Debt”, another account “Rental Debt” must be used. The change must impact at definition level, because new renter payment entries must be done in accordance to the new definition. But at the same time, old “renter payment” entries must continue being identified (and classified) as such, even though they do not follow the new definition. Indeed it is possible to add an attribute with two allowed values (current/not current) to the table DetDefs, to differentiate current detail definitions from not current ones. Hence, accounting entries done according to old definitions can also be identified as “Renter payment” entries.

Req. 11. Differenciate definitional and operational views as two abstraction levels. The previous requirement shows the benefits of having definitional and operational levels in separate structures. Despite this, it is hidden in the application code the fact that tables EntryDefs and Entries keep semantically equivalent information at different levels. Besides a greater expressiveness, a model with the capability to represent that EntryDefs, DetDefs and Accounts are in the definitional level whereas Entries and Dets are in the operational level, avoids errors of design. For example, at the moment of reusing SIGGA relational schema, it is useful to distinguish a FK from Entries to EntryDefs (that links conceptually equivalent knowledge at different levels) from a FK from DetDefs to Accounts (that links two conceptually different objects at the same level).

The basic requirements which are Req. 1 to Req. 6 come from the ALE-based accounting model. However, only Req. 2 to Req. 6 can be verified by the scheme shown in Figure 1. The schema also verifies Req. 10. Req. 1 and Req. 7 are verified by non-structural restrictions (in the code). Req. 8, Req. 9 and Req. 11 are desirable constraints for the implementation of the SIGGA application that could be verified by non-structural restrictions but they are not satisfied at the moment.

3 Ontological modelling

As discussed in the previous section, there are some requirements in the SIGGA accounting module that need to be represented through classification criteria. These criteria varies from the simplest case of Req. 2 where accounting details can be credits or debits to the most complex classifications from Req. 6 or Req. 9, for example. Traditionally, ontologies are proved to be a good tool for represent taxonomies, and moreover to model relations that hold between taxonomy classes. The SIGGA application needs not only model clasification criteria among accounting details but also model relations between classes of accounting details as described in Req. 9, 10 and 11. In this section we present an ontological model for the SIGGA accounting module and discuss the capabilities for requirements satisfiability that this model has. First we propose a model in OWL2 language [6], and then we propose a model in OWL2 extended with meta-modelling. Finally, we compare pros and cons of both approaches.

3.1 The SIGGA ontological model (OM)

In this section we introduce the SIGGA ontological model (OM) which captures the classes (concepts) and properties (roles) that experts need to define. This is depicted in Figure 2 where ovals represent classes and arrows represents properties. Table 1 shows some of the Tbox and Rbox axioms of OM. The class *Account* represents different classes of accounts, i.e. Assets, Availability, etc. The class *DC* represents the two possibilities of “credit” or “debit”. The class *Entry* represents the grouping of entries by the definition type, i.e. the set of entries of type “Renter payment”, the set of entries of type “Monthly calculation of rent”, etc. These are disjoint sets of entries. The properties *detailD* and *detailC* have domain *Entry* and range *Det*. They are introduced to distinguish between details that correspond to “debit” and “credit”, and have a super-property *detail*, which is the disjoint union of *detailD* and *detailC*, defined conveniently to express some restrictions common to both properties. For the sake of clarity, the class *D/C* is defined with the instances *debit* and *credit*. The role *hasDC* connects details at debit or credit with the corresponding instances of *D/C*. The property *account* connects entry details to accounts. Note that axioms (6) to (8) show an example for the subclass *RenterPayEnt* of *Entry* and the corresponding subclasses of *Det* connected by *detailD* and *detailC*. The same axioms must be declared for all subclasses of *Entry*.
Axioms (6) to (8) represent the definitional level because they define what accounts must be at debit and credit for each kind of accounting entry.

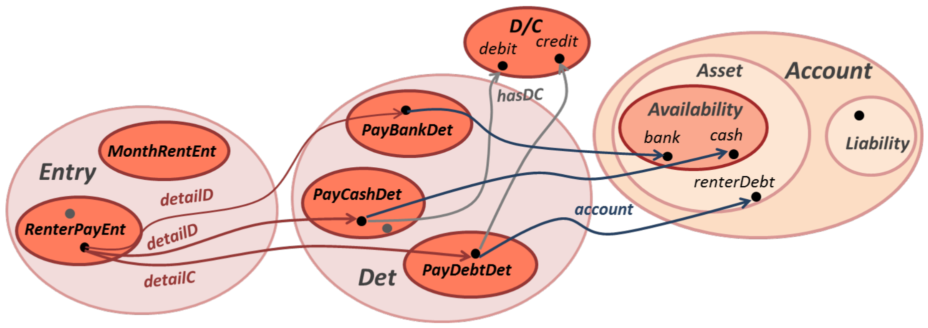


Fig. 2: Example of SIGGA OM design

Axiom	Description
(1) $Det \sqsubseteq = 1 detail^{-}.Entry$	A detail corresponds to exactly only one accounting entry.
(2) $Entry \sqsubseteq \exists detailD.\top \sqcap \exists detailC.\top$	Entries are balanced double entry records.
(3) $detailD \sqsubseteq detail$ $detailC \sqsubseteq detail$ $Dis(detailD, detailC)$ $Entry \sqsubseteq \forall detail.(\exists detailD^{-}.Entry \sqcup$ $\exists detailC^{-}.Entry)$	*detail* is the disjoint union of *detailD* and *detailC* .
(4) $\top \sqsubseteq \forall detailD.(\exists hasDC.\{debit\} \sqcap$ $\forall hasDC.\{debit\})$ $\top \sqsubseteq \forall detailC.(\exists hasDC.\{credit\} \sqcap$ $\forall hasDC.\{credit\})$	Roles *detailD* and *detailC* are associated to debit and credit.
(5) $Det \sqsubseteq = 1 account.Account$	A detail has associated a single account.
(6) $RenterPayEnt \sqsubseteq \forall detailD.$ $(PayCashDet \sqcup$ $PayBankDet)$ $RenterPayEnt \sqsubseteq \forall detailC.PayDebtDet$	Subclasses of *Entry* are described by the subclasses of Det they have associated at debit and credit
(7) $PayCashDet \sqsubseteq \forall detail^{-}.RenterPayEnt$ $PayBankDet \sqsubseteq \forall detail^{-}.RenterPayEnt$ $PayDebtDet \sqsubseteq \forall detail^{-}.RenterPayEnt$	Subclasses of Det correspond to only one subclass of *Entry*.
(8) $PayCashDet \sqsubseteq \exists account.\{cash\}$ $PayBankDet \sqsubseteq \exists account.\{bank\}$ $PayDebtDet \sqsubseteq \exists account.\{renterDebt\}$	Subclasses of Det are described by the accounts they have associated at debit and credit.

Table 1: Example of SIGGA OM Tbox and RBox

3.2 The SIGGA Ontological Meta Modelling (OMM)

The SIGGA ontological model (OM) depicted in the previous section does not represent explicitly the relations that hold between definitional and operational level of the SIGGA accounting module presented in Section 2. In order to capture these relations we extend OM with another ontology (the definitional ontology) illustrated on the top of Figure 3, that captures the definitional level. At this level we have a class *EntryDef* that represents the set of accounting entry denitions, a class *DetDef* that represents how debit and credit details must be registered for each entry denition, and also classes *Account* and *D/C* (as in OM). The role *detailDef* has domain *EntryDef* and range *DetDef*, it is a superrole of *detailDefD* and *detailDefC* that distinguish debit from credit detail definitions. At the lower part of the figure it is depicted a subschema of OM which represents uniquely entries and details at operational level. It is the ontology in the OM design, without the classes *Account* and *D/C*. Table 2 shows some of the Tbox and Rbox axioms of OMM, where axioms (1) to (3) are the same as in OM, and also Mbox axioms with a new constructor we describe below.

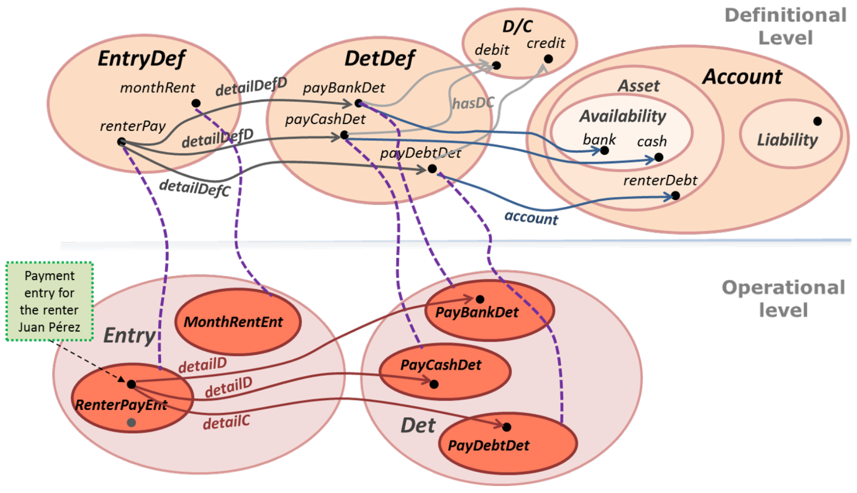


Fig. 3: Example of SIGGA OMM design

In the definitional ontology, each kind of accounting entry is represented as an instance of *EntryDef* whereas in the operational ontology each kind of entry is represented as a class which is a subclass of *Entry*, and agrees with a single entry definition in *EntryDef*. For instance, renter payments are represented at definitional level with the individual *renterPay* but at operational level with the class *RenterPayEnt*. Instances of *RenterPayEnt* are all different, such as the “Juan Pérez payment for $5,000” and the “Pedro Vidal payment for $8,000”, but all of them have debit and credit details which agree with detail definitions associated (by roles *detailDefD* and *detailDefC*) to the individual *renterPay*. The fact that the individual *renterPay* is semantically equivalent to the class *RenterPayEnt* is represented in Figure 3 by a dotted line that connects them. This means that the interpretation of the individual *renterPay* must be equal to the interpretation of the class *RenterPayEnt*. For this, we introduce the axiom $renterPay =_{\mathsf{m}} RenterPayEnt$ to equate the individual *renterPay* with the class *RenterPayEnt* [13]. Similar equality axioms are added for *monthRent* and *EntryMonthRent*, *payCashDet* and *PayCashDet*, and so on. In that way, each ontology conceptualizes the same business objects at different knowledge level.

Axiom	Description
(1) $Det \sqsubseteq = 1 detail^{-}.Entry$	A detail corresponds to exactly only one accounting entry.
(2) $Entry \sqsubseteq \exists detailD.\top \sqcap \exists detailC.\top$	Entries are balanced double entry records.
$detailD \sqsubseteq detail$ $detailC \sqsubseteq detail$ (3) $Dis(detailD, detailC)$ $Entry \sqsubseteq \forall detail.(\exists detailD^{-}.Entry \sqcup$ $\exists detailC^{-}.Entry)$	*detail* is the disjoint union of *detailD* and *detailC*
(4) $DetDef \sqsubseteq = 1 detailDef^{-}.EntryDef$	A detail definition corresponds to exactly only one accounting entry definition.
$detailDefD \sqsubseteq detailDef$ $detailDefC \sqsubseteq detailDef$ (5) $Dis(detailDefD, detailDefC)$ $EntryDef \sqsubseteq \forall detailDef.$ $(\exists detailDefD^{-}.EntryDef \sqcup$ $\exists detailDefC^{-}.EntryDef)$	*detailDef* is the disjoint union of *detailDefD* and *detailDefC*
$\top \sqsubseteq \forall detailDefD.(\exists hasDC.\{debit\} \sqcap$ (6) $\forall hasDC.\{debit\})$ $\top \sqsubseteq \forall detailDefC.(\exists hasDC.\{credit\} \sqcap$ $\forall hasDC.\{credit\})$	Roles *detailDefD* and *detailDefC* are associated to debit and credit.
(7) $EntryDef \sqsubseteq \exists detailDefD.\top \sqcap$ $\exists detailDefC.\top$	Accounting entry definitions are balanced double entry records.
(8) $DetDef \sqsubseteq = 1 account.Account$	A detail definition has associated a single account.
(9) $\mathsf{MetaRule2}(detailDefD, detailD)$ $\mathsf{MetaRule2}(detailDefC, detailC)$	Mbox axioms introducing sets of Tbox axioms that follow certain pattern.

Table 2: SIGGA OMM Tbox, Rbox and Mbox.

From [13] we recall that *an ontology $\mathcal{O}$ with meta-modelling* is a tuple $(\mathcal{T}, \mathcal{R}, \mathcal{A}, \mathcal{M})$ where $\mathcal{T}$, $\mathcal{R}$ and $\mathcal{A}$ are a Tbox, Rbox and Abox respectively, and $\mathcal{M}$ is an Mbox, which is a set of meta-modelling axioms (such as $renterPay =_{\mathsf{m}} RenterPayEnt$). We now extend the Mbox by adding MetaRule and MetaRule2 axioms to $\mathcal{M}$. We also recall that an *interpretation $\mathcal{I}$ of and ontology $\mathcal{O} = (\mathcal{T}, \mathcal{R}, \mathcal{A}, \mathcal{M})$* has a domain of interpretation Δ that can contain sets, sets of sets, and so on, since with meta-modelling an individual

(equated to a class) can be interpreted as a set.
Moreover, in order to explicitly declare correspondences established between definitional level roles and operational level roles, in the following, we introduce introduce the new constructor $\mathsf{MetaRule}(R, S)$ for roles R and S. The intuition behind the new constructor is that pairs (a, b) in R in the higher level are translated as TBox axioms with S in the lower level. For convenience, we also introduce the constuctor $\mathsf{MetaRule2}$. We say that $\mathcal{I} \models \mathsf{MetaRule}(R, S)$ if $A^{\mathcal{I}} \subseteq (\forall S.(\sqcup X))^{\mathcal{I}}$ where $X = \{B \mid (a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$ and $a^{\mathcal{I}} = A^{\mathcal{I}}$ and $b^{\mathcal{I}} = B^{\mathcal{I}}\}$ (note that $\sqcup\emptyset = \top$). We say that $\mathcal{I} \models \mathsf{MetaRule2}(R, S)$ iff $\mathcal{I} \models \mathsf{MetaRule}(R, S)$ and $\mathcal{I} \models \mathsf{MetaRule}(R^{-}, S^{-})$.
Note that OMM contains the axioms (9) expressed using $\mathsf{MetaRule2}$ which is equivalent to adding a whole set of axioms following a certain pattern such as (6) and (7) of OM.

From [13] we recall that the tableau algorithm for checking consistency of an ontology with meta-modelling extends the tableau algorithm for an ontology without meta-modelling by adding three expansion rules (to transfer equalities and inequalities between individuals with meta-modelling to corresponding concepts) and a condition to avoid the domain to be a non well-founded set. To deal with the $\mathsf{MetaRule2}$ constructor, we have to add the new expansion rules to the algorithm, which exceeds the scope of the present work.

3.3 Comparing the two ontological models

In this section we analyze how OMM design accomplishes the set of requirements presented in Section 2 highlighting the pros and cons of this approach. We compare the OM and the OMM designs presented in previous section.
Req. 1 is solved in OM and OMM the axiom (2) in both tables 1 and 2.
Req. 2 is solved in OM and OMM by the axiom (3) in both tables 1 and 2..
Req. 3 is solved in OM by the axiom (5) of Table 1 whereas for OMM by the axioms (8) and (9) of Table 2.
Req. 4 is solved in OM and OMM by the axiom (1) in both tables 1 and 2.
Req. 5 is solved for OM by axioms such as Eq. (7) in Table 1 that must be declared for each subclass of *Det*. However, OMM solves it by the axiom (4) in Table 2.
Req. 6 is solved for OM by axioms (6) and (7) in Table 1 but OM has several drawbacks: i) there is no definitional level to verify the correspondence, and ii) the axioms must be declared for each kind of accounting entry. This is a relevant aspect because the real system has defined around 80 different kinds of accounting entries, and this amount is still growing. For OMM it is not needed to express it for each entry definition. The requirement is ensured by combining meta-modelling axioms (that equate instances of *EntryDef* to subclasses of *Entry* and instances of *DetDef* to subclasses of *Det*) with the $\mathsf{MetaRule2}$ constructor (axioms (9) in Table 2). Whereas for OM entry definitions are expressed in the TBox, for OMM they are registered in the ABox as “data”, so OMM is more flexible for dynamic business rules.
Regarding Req. 7, both OM and OMM have the capability of classifying entries by a kind of accounting entry, since this is given by the subclasses of *Entry*. If we consider another relevant criteria such as “all entries with the account Cash at debit”, for the first

design we just introduce the class $\exists detailD.(\exists account.\{cash\})$. For OMM it is not so direct, because we have accounts at definitional level. We could solve it if we could express higher order queries:

$$q(y) = \exists X\ \exists x\ \exists z.account(x, bank) \wedge x =_{\mathsf{m}} X \wedge X(z) \wedge detailD(y, z) \tag{1}$$

Req. 8 is about classifying entry definitions according to a given criteria, for instance "all entry definitions with the account Bank at debit". For OM, we cannot classify entry definitions since they are not represented as instances, they are classes. Hence, it would be necessary to explore the TBox axioms of the form of (8) in Table 1 looking for classes with the text "bank" in their definition, and then to obtain classes of the form of (6) in Table 1. Besides being a mechanism a bit cumbersome, the classification is not expressed in the model at all. However, for the OMM design the requirement is solved by introducing the class $\exists detailDefD.(\exists account.\{bank\})$.

To analize Req. 9 we consider the example presented in Section 2, in which a given renter cannot pay his damage expenses because the debt account was not included in the "renter payment" entry definition. Again, for OM, to validate that all asset accounts that generate debts are in the "renter payment" entry definition as credits, we would have to explore all axioms of the form (8) in Table 1. The problem is that the requirement is about validating definitions, not "data". By contrast, with OMM we solve the requirement just adding the following axiom:

$$\begin{aligned} &Asset \sqcap \neg Availability \sqcap \exists account^{-}.(\exists detailDefD^{-}.\top) \sqsubseteq \\ &\exists account.^{-}.(\exists detailDefC^{-}.\{renterPay\}) \end{aligned} \tag{2}$$

Another benefit of validating definitions, is that for certain kind of requirements we can check for a condition only once over an accounting entry definition instead of checking the same condition over all concrete accounting entries of that definition. For instance, for all entries that move availability accounts (cash, bank), it must hold that, if an availability account is at debit, no non availability account can be at debit (and the same at credit). This condition can be easily checked for the OM and OMM designs by introducing respectively the following axioms:

$$\begin{aligned} &Entry \sqcap \exists detailD.(\exists account.Availability) \sqsubseteq \\ &Entry \sqcap \forall detailD.(\exists account.Availability) \end{aligned} \tag{3}$$

$$\begin{aligned} &EntryDef \sqcap \exists detailDefD.(\exists account.Availability) \sqsubseteq \\ &EntryDef \sqcap \forall detailDefD.(\exists account.Availability) \end{aligned} \tag{4}$$

But for OM, all daily transactions such as those of renter payment (that moves cash or bank) must be validated, and the same happens for other conditions. However, with OMM it is enough to check these conditions at the moment of introducing each definition because the MetaRule2 ensures that concrete entries agree with corresponding definitions. This requirement is relevant since it shows the role played by the definitional ontology in the OMM design. The fact that there is an ontology that "describes definitions" turns out natural defining relations and imposing restrictions between them. Thus, experts can check for inconsistencies or inferences in their definitions before the application is running. In the example of Eq. (2), if some instance of *Asset* at debit in some entry definition (left part of the axiom) is not associated to the "renter payment" entry definition at credit (right part of the axiom), then the reasoner will create the association.

We now discuss Req. 10. Consider the example of Section 2 and suppose an expert change the definition of the "Renter Payment" entry. Instead of the account "Renter Debt", a new account "Rental Debt" must be at credit, but old entries must be kept unchanged and classified as "Renter Payment" entries. For the OM design, axioms (6) in Table 1 should be modified as follows:

$$\begin{aligned} RenterPayEnt &\sqsubseteq \forall detailD.(PayCashDet \sqcup PayBankDet) \\ RenterPayEnt &\sqsubseteq \forall detailC.(PayDebtDet \sqcup PayRentDet) \end{aligned} \tag{5}$$

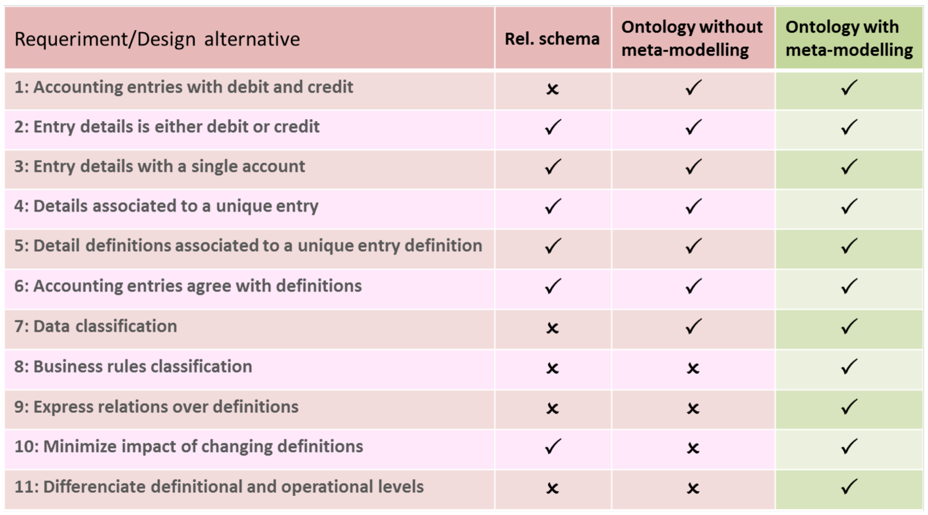
Moreover, in (7) and (8) in Table 1 the axioms below should be added:

$$\begin{aligned} PayRentDet &\sqsubseteq \forall detail^{-}.RenterPayEnt \\ PayRentDet &\sqsubseteq \exists account.\{rentalDebt\} \end{aligned} \tag{6}$$

The problem with this new "Renter Payment" entry definition is that it is not clear what is the current definition. A solution to differenciate accounting entries that agree with the current definition from those that do not, could be to define a new class *State* with two instances *current* and *nonCurrent*, and associate instances of *Det* to one of these states. But as all entries with the "Renter Debt" account associated are deprecated, we have a redundance since we represent definitional aspects at operational level. Another solution is to define a subclass *CurrentRenterPayEnt* of *RenterPayEnt*, but we have to define these subclasses for all subclasses of *Entry*. For OMM we can associate *current* or *nonCurrent* to instances of *DetDef* at definitional level. So, we can explicit what detail definitions are current and what are not.
Finally, we analize Req. 11. Regarding the OM design, the expert's view is represented by TBox axioms (6), (7) and (8) in Table 1 that describe different subclasses of *Entry*, and the operator's view by ABox axioms which assert to what kind of accounting entry (a subclass of *Entry*) concrete entries belong. However, TBox axioms (6), (7) and (8) in Table 1 indeed describe business "data" but not business rules (definitions). So, in practice OM properly conceptualize relations and restrictions at the operational level whereas provide a weak conceptualization at definitional level. By contrast, for the OMM design there is a definitional ontology (upper part of Figure 3) that describes business rules, allows to introduce relations between definitions and is conceptually uncoupled from the ontology (lower part of Figure 3) that describes relations over business "data" (operational level). Hence, the OMM design fully fulfills the requirement.

4 Conclusions and future work

In this work we present how an ontological meta-modeling approach can be applied to a real case on the accounting domain based on the ALE system. The ontoREA ontology [5] conceptualizes business transactions of the ALE system. However, ontoREA models the ALE system from a unique point of view. Our approach subsumes ontoREA, we use it at definitional level as vocabulary to model the accounting concepts from the ALE system. In this work we show practical reasons that the ontological meta-modelling here presented provides a better expressivity level than both the relational model and the ontology model without meta-modelling. Table 3 summarizes advantages and drawbacks for each design alternative, illustrating how the set of requirements of the case study is expressed in the relational model, and in the two ontology-based designs.



Requeriment/Design alternative	Rel. schema	Ontology without meta-modelling	Ontology with meta-modelling
1: Accounting entries with debit and credit	×	✓	✓
2: Entry details is either debit or credit	✓	✓	✓
3: Entry details with a single account	✓	✓	✓
4: Details associated to a unique entry	✓	✓	✓
5: Detail definitions associated to a unique entry definition	✓	✓	✓
6: Accounting entries agree with definitions	✓	✓	✓
7: Data classification	×	✓	✓
8: Business rules classification	×	×	✓
9: Express relations over definitions	×	×	✓
10: Minimize impact of changing definitions	✓	×	✓
11: Differenciate definitional and operational levels	×	×	✓

Table 3: Expressivity achieved in the SIGGA accounting module with Relational Scheme, Ontological design (OM) and Ontological Meta-Modelling design (OMM)

For the OMM approach, the model itself provide elements to fully represent the set of requirements. Among them, the most relevant are the capabililty for checking the correctness of expert definitions as well as dealing with the impact of changing them, what is achieved since definitions are for experts "data" that they can manage and even modify if changes in the business require it.
Our meta-modelling approach is also useful for other application domains. The example of page 28 in [4] can easily be represented with OMM, with the axioms below:

$$\begin{array}{lll} ProcModel(A6) & MobileModel(IPhone5) & compatible(A6, IPhone5) \\ A6 \sqsubseteq Processor & IPhone5 \sqsubseteq Mobile & \\ \exists compatible.\top \sqsubseteq ProcModel & \top \sqsubseteq \forall compatible.MobileModel & \\ \exists installedIn.\top \sqsubseteq Processor & \top \sqsubseteq \forall installedIn.Mobile & \end{array} \tag{7}$$

We see that $A6$ and $IPhone5$ have an implicit meta-modelling since they are individuals and also classes. Moreover, $installedIn$ is a role between superclasses of $A6$ and $IPhone5$ whereas $compatible$ is a role between metaclasses $ProcModel$ and $MobileModel$. By introducing the axiom:

$$\mathsf{MetaRule2}(compatible, installedIn) \tag{8}$$

we ensure that instances of $A6$ will be linked to instances of $IPhone5$ through the role $installedIn$.
As a general conclusion, the ontological meta-modeling approach (OMM) for modelling a business application provides benefits from the perspective of different user

profiles. From the designer/developer's perspective, the approach results profitable both in the design and in the maintenance of the application. In the application design, it provides an intuitive and flexible way for introducing business rules at different abstraction levels. In the application maintenance, having business rules expressed as axioms helps prevent mistakes, which can happen when changing code. From the expert user's perspective, it is a useful mechanism to validate the correctness of their definitions since it allows to relate definitions as well as declare restrictions over them. From the operator's perspective, as the correctness of the definitions can be validated, a lot of problems are avoided at "application operation time" during customer service.
In the near future, we plan to investigate algorithms for checking consistency of ontologies with the new constructor MetaRule2 and for answering higher order queries such as Eq. (1). Moreover, we also plan to enhance a prototype of the accounting case study implemented over a relational database, by adding the implemented ontologies with meta-modelling as detailed in the present work.

References

1. A. Abbasi and N. Kulathuramaiyer. A systematic mapping study of database resources to ontology via reverse engineering. *Asian J. Inform. Technol.*, 15, 2016.
2. C. Atkinson and T. Kühne. In defence of deep modelling. *Information & Software Technology*, 64, 2015.
3. C. Atkinson, T. Kühne, and J. de Lara. Editorial to the theme issue on multi-level modeling. *Software and System Modeling*, 17, 2018.
4. V. A. de Carvalho and J. P. A. Almeida. Toward a well-founded theory for multi-level conceptual modeling. *Software and System Modeling*, 17, 2018.
5. C. Fischer-Pauzenberger and W. S. A. Schwaiger. The OntoREA Accounting Model: Ontology-based modeling of the accounting domain. *CSIMQ*, 11, 2017.
6. B. C. Grau, I. Horrocks, B. Motik, B. Parsia, P. F. Patel-Schneider, and U. Sattler. OWL 2: The next step for OWL. *J. Web Sem.*, 6, 2008.
7. N. Guarino. Some ontological principles for designing upper level lexical resources. *CoRR*, cmp-lg/9809002, 1998.
8. G. Guizzardi. Ontological foundations for conceptual modeling with applications. In *Advanced Information Systems Engineering. CAiSE 2012*, 2012.
9. G. Guizzardi and G. Wagner. A unified foundational ontology and some applications of it in business modeling. In *CAiSE'04 Workshops in connection with The 16th Conference on Advanced Information Systems Engineering, Knowledge and Model Driven Information Systems Engineering for Networked Organisations*, 2004.
10. M. Hazber, R. Li, X. Gu, and G. Xu. Integration mapping rules: Transforming relational database to semantic web ontology. 10, 2016.
11. M. Lenzerini, L. Lepore, and A. Poggi. Metaquerying made practical for OWL 2 QL ontologies. *Information Systems*, 2018.
12. W. B. Meigs and R. F. Meigs. *Financial accounting*. McGraw-Hill, 4th ed. edition, 1983.
13. R. Motz, E. Rohrer, and P. Severi. The description logic *SHIQ* with a flexible meta-modelling hierarchy. *J. Web Sem.*, 35(4).
14. K. Munir and M. S. Anjum. The use of ontologies for effective knowledge modelling and information retrieval. *Applied Computing and Informatics*, 2017.
15. A. Olivé. The universal ontology: A vision for conceptual modeling and the semantic web (invited paper). In *Conceptual Modeling - 36th International Conference*, 2017.