REA-DSL: Business Model Driven Data-Engineering

Dieter Mayrhofer, Christian Huemer
Institute of Software Technology and Interactive Sytems
Vienna University of Technology
Vienna, Austria
Email: {lastname}@big.tuwien.ac.at

***Abstract*—An accounting information system (AIS) manages data about a company's financial and economic status. The contribution of this paper is closing the gap between the languages used by business domain experts and IT-experts in analyzing the relevant data. A well accepted approach scrutinizing an accountability infrastructure is the Resource-Event-Agent (REA) ontology. Although REA has been based on well-established concepts of the accounting theory, its representation has not been intuitive to domain experts. In previous work, we developed the REA-DSL, a dedicated and easy-to-understand graphical domain specific modeling language for the REA ontology. Evidently, a model-driven approach requires to transform the REA-DSL artifacts to code. In this paper we present the transformation of the REA-DSL to a relational database for AIS. This approach offers the advantage that a domain expert verifies the relevant data in an "accounting language", whereas the IT expert is able to work with traditional data base structures.**

I. Introduction

Accounting information systems (AIS) provide business professionals with accurate and reliable information about a company's financial and economic status. Furthermore, it allows them to predict future cash flows and workloads and, consequently, make decisions and take aligned actions. According to Romney and Steinbart, an AIS is a system that collects, records, stores, and processes financial and accounting data to produce information for decision makers [1].

When designing an AIS, it is essential that the underlying data structure and the user interface reflect the economic phenomenas a company is based on. Thus, it is crucial that business professionals who have the knowledge about these economic phenomenas can unambiguously communicate the requirements with the IT professionals, who are in charge of designing and developing the AIS. Business ontologies can serve as an unambiguous communication language by providing a set of concepts and relations between them for describing businesses and their economic actions.

The most prominent and well-accepted business ontology language in the area of AIS is the Resource-Event-Agent (REA) ontology developed by McCarthy and Geerts [2], which is based on well established concepts of the economic literature. It is based on the modeling of economic events on the operational layer that result in an increment or decrement of economic resources and economic agents carrying out these events. Over the years REA has been extended by new concepts – the planning and policy layer [3]. These layers provide concepts to model commitments on events that are planned for the future. Nowadays, REA is considered as a sophisticated business modeling language for developing an AIS. However, REA does not come with a dedicated graphical syntax, rather it uses stereotyped class-diagrams that are not intuitive to business users. Furthermore, REA misses an unambiguous formalization of the class diagrams, since the stereotypes and their relations are not defined by a UML profile. We argue, that these limitations diminish the use of REA and that REA could greatly benefit from a dedicated graphical representation like e3-value [4] – another language for business modeling with a dedicated focus on value exchanges – does.

Consequently, we took the challenge to create an unambiguous and easy to understand domain specific language for REA called the REA-DSL. In previous publications [5], [6] we have formalized the REA-DSL by means of an underlying meta model and an intuitive graphical representation with dedicated stencils for all REA concepts. However, in its current state the REA-DSL still misses indispensable concepts required for the design of an AIS database scheme – i.e. properties and primary keys. Consequently, we first extend the REA-DSL by properties and primary keys.

Given these extensions, the REA-DSL is promoted to a tool supporting the business experts and IT experts in capturing the requirements for an AIS. Nevertheless, these models cannot be read and processed by a database system. The IT professional would still be required to manually transform the REA-DSL to a relational model adhering to the REA ontology rules. This task is error prone and time intensive – resulting in additional costs for designing the AIS. Consequently, in the second part of this paper we present a mapping between the REA-DSL and a corresponding relation model. Similar to the development of REA itself, we focus in this work on mapping the REA operational layer and consider the planning and policy layer mapping for future work. Additionally, our REA-DSL tool implementation provides an automatic generation of a relational model from the REA-DSL based on the proposed mapping. This leads to a model-driven approach transforming the REA-DSL artifacts to code. This approach offers the advantage that a domain expert verifies the relevant data in an "accounting language", i.e. the REA-DSL, whereas the IT expert is able to work with traditional data base structures.

The remainder of the paper is structured as follows: In

Section II we present related work on REA. We illustrate our REA-DSL by means of an example in Section III. Afterwards, we introduce the extension of the REA-DSL by properties and primary keys in Section IV. The core of the paper – Section V – elaborates on the mapping rules between the REA-DSL and the relational model. An evaluation is provided in Section VI. A summary in Section VII concludes the paper.

II. Related work

In its beginnings REA was developed by William McCarthy as an accounting framework where economic exchanges are the central idea [2]. According to the acronym REA, the main concepts are *economic resources*, *economic events*, and *economic agents*. For readability issues we will drop the prefix *economic* for the remainder of this paper. Basically, one or more resources are exchanged between usually two (but in theory also more) agents at well defined events. A cornerstone of REA is also the concept of duality, which means that usually one event (or in theory a set of events) is compensated by another event (or set of events). An example on the instance level may be: On the 19 March 2012 a `sale` (*event*) occurs, where the `salesman Joe` (*agent*) with the help of the `shop assistants Mary` and `Wendy` (*agents*) give `50 pounds of tuna fish` (*resource*) and a `fishing rod` (*resource*) to their `customer Fred` (*agent*). The `sale` (*event*) is compensated by the `payment` (*event*) which happens right after `Fred` (*agent*) pays the amount of `700 Euros` to the `cashier Mark` (*agent*).

Evidently, REA does not model the individual instance on M0, but analyses an enterprise on the model layer M1. Accordingly, the REA model defines the schema for the above mentioned instance from the perspective of the `seller` as follows: `sales` and `payments` are in *duality*. A `sale` is performed by exactly one `salesman`, a number of `shop assistants` and an external `customer`. The `sale` leads to a decrease of `fish` and additional `products`. The `sale` is compensated by the increase in `cash` being part of the `payment` which takes place between an external `customer` and a `cashier`. In order to build this REA model, REA provides the concepts of *Resources*, *Events* and *Agents* on the meta model layer M2. Furthermore, the M2 layer covers the concepts of *duality* (relationship between events), *stock-flows* (relationship between event and resource) and *participate* (relationship between event and agent).

In an AIS it is not only important to capture what is happening or has happened, it is also important to record what is planned or what contracts were made in order to plan and predict future events which affect resources and agents. Consequently, REA was extended by the *planning* and *policy layer* [3], [7], [8] which allows to model *policies* and *commitments* for the future events and typification of resources, events, and agents. Types are very important for the planning and policy layer but are also used by the operational layer. In this work we focus on the operational layer mentioned above and also consider typification concepts.

To extend the REA-DSL with properties and primary keys we make use of common modeling practice in Entity-Relationship (ER) modeling and refer to basic database literature [9], [10]. As for the mapping from the REA-DSL to the relational model we also consulted the aforementioned books as well as the hierarchy mapping strategies in [11]. In [12] Poels compares REA with entity relationship (ER) diagrams in terms of conceptual modeling of AIS. The empirical study focuses on the perspective of the users who need to use and understand the conceptual REA models (e.g. database designers, system developers, information analysts, and AIS end-users) and showed the effectiveness of REA modeling in the area of AIS. However, we showed in [6] that our REA-DSL was even more intuitive and easier to comprehend than the original class based REA diagrams.

III. REA-DSL Example

To understand the mapping between the REA-DSL and the relational model we first provide a simple example of the REA-DSL for the reader to get familiar with the main concepts. The interested reader on the underlying meta-model of our REA-DSL is referred to our publication [5]. On the left side of Figure 1 an example of a value chain view for a fish sale company called Sy's Fish is depicted. This example has also been used by McCarthy and Geerts [13]. A value chain specifies the flow of resources between value activities. A value activity is an activity that takes input resources to create output resources that are by economic principles of higher value to the company. Each value activity is further detailed by a REA duality relationship as described in the previous section.

The left hand side of Figure 1 shows the value chain of Sy's Fish. It covers the value activities: product buying, fish buying, transport, truck acquisition, cleaning, selling, and payroll. Resources - depicted by a drop - are usually created by one value activity and may serve as input to other value activities. For example, selling creates cash that is used as input to most of the other activities.

Each of the value activities is refined by a *duality* model. The duality model defines what events "are happening" or "have happened" and thus allows to model in detail what agents and resources are involved in economic events. On the right side of Figure 1 we picked the duality model of the `selling` value activity as an example. It defines, that in an event `sale` (depicted by a hexagon) an inside agent `salesman` (depicted by a white stick figure) and multiple `shop assistants` (depicted by a stack of stick figures) sell the bulk resources `fish` (depicted by a dashed drop) and the regular resource `products` (depicted by a solid drop) to an outside agent `customer` (depicted by a black stick figure). Regular resources (products) are uniquely identifiable whereas bulk resources (fish) are not uniquely identifiable and only the amount on hand can be stored. In return for the `sale` event – according to the "give and take" principle of economic exchanges – the `customer` gives `cash` to the `cashier` in the `payment` event.

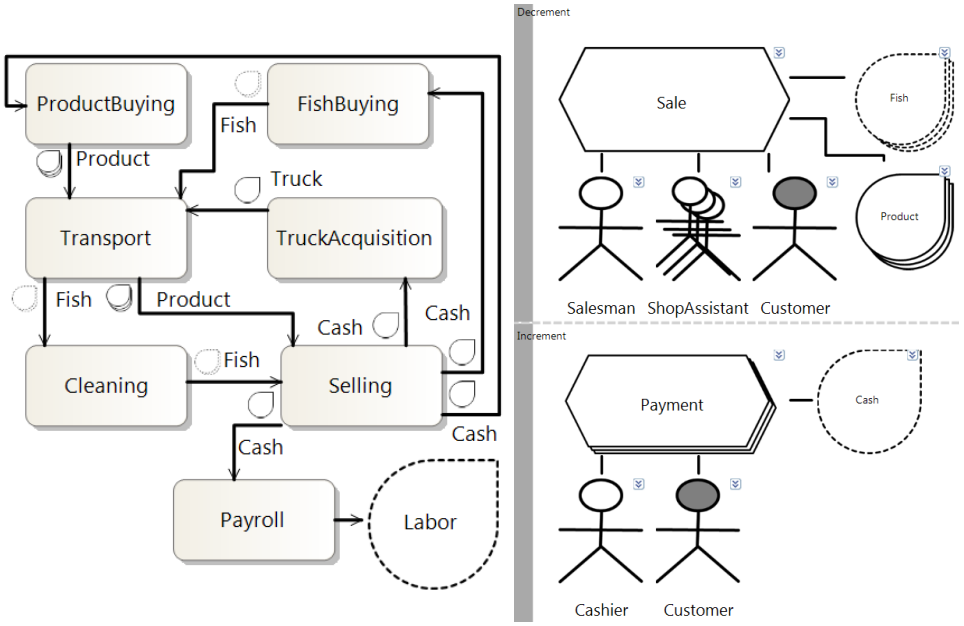

Fig. 1. REA example

IV. REA-DSL Extended: Adding Properties and Keys

In order to derive relational models from a REA-DSL model, we first extend the main REA-DSL concepts agents, resources, and events by properties and primary keys. In the following, we use the concepts of resource and resource type to demonstrate the extension by primary keys and properties. The extension for agents and events follows the same principles. Figure 2 shows the identifiable resource `product` and the bulk resource `fish` that are compliant to the meta model in the middle. The numbers in the description below reference the numbers in Figure 2.

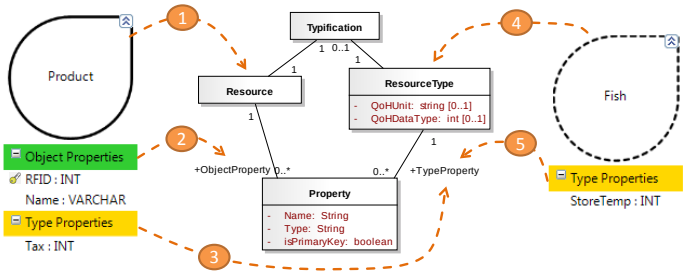

Fig. 2. Resource/Bulk Resource Properties Meta-Model

The identifiable resource (1) on the left side is depicted by a drop and contains two compartments: on top the `object properties` compartment colored in green (2) and at the bottom the `type properties` compartment colored in yellow (3). These compartments can contain multiple properties. The instances of the object properties vary between each individual product. However, there are also static properties which stay the same for certain types of products. Thus, REA distinguishes between *resources* with object properties and *resource types* with type properties. For example, all product types *book* or all product types *fishing rods* have the same common *tax percentage*. On the other hand, each single resource `book` or resource `fishing rod` has its own unique `RFID` code. Therefore, the meta model specifies, that one resource has exactly one *typification* relationship to exactly one *resource type* and vice versa. Consequently, there is a one-to-one relationship between the resource and the resource type which are depicted by a common dashed drop in the concrete syntax (1,3). In the object-oriented paradigm classes may include static attributes (class attributes) and non-static (regular) attributes. In the REA context it is required to have one concept covering the regular attributes (*object properties*) and another one covering the static attributes (*type properties*). For example, an order for a car might just reference the resource type (e.g., a Ferrari Type F40) containing the type properties or the exact individual resource (e.g., the Ferrari F40 with the serial number `XYZ10` and golden painting) containing the object properties.

In our example, the resource `product` contains two object properties `RFID:INT` and `Name:VARCHAR` as well as one type property `Tax:INT`. Properties consist of a *name* and a *type*. In the concrete syntax on the left side the name is separated from the type by a colon (e.g. `NAME:VARCHAR`). The flag *isPrimaryKey* signals, whether or not this property is a unique identifier of the resource. Primary key properties are tagged by a key symbol. In our example `RFID:INT` is the primary key property for the resource product. The type property `Tax:INT` applies to certain types of products. Consequently, a certain type of product (e.g., books) will always have the same tax applied on it (e.g., 7 %).

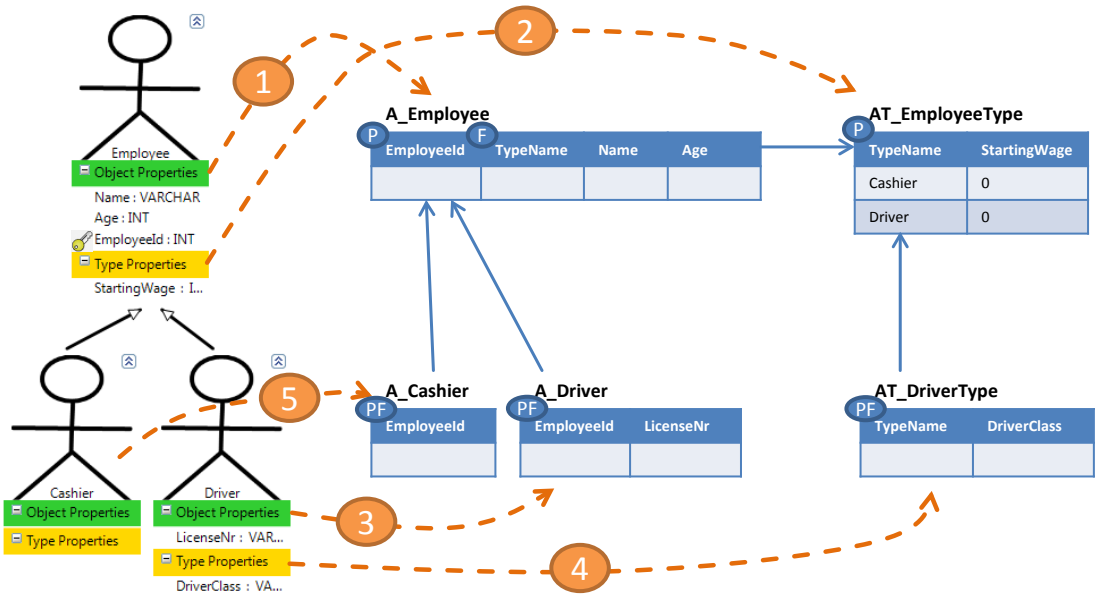

Fig. 3. Agent Mapping

On the right side of Figure 2 a bulk resource (resource type) `fish` (4) is depicted. A bulk resource cannot be individually identified. In our `fish` case, we can only record the amount of a type of `fish` but cannot track each individual `fish`. Thus, it is not possible to apply object properties for an individual `fish`. Consequently, bulk resources may only contain type properties (5). The `fish` bulk resource contains the type property `StoreTemp:INT`. You can now record for each type of `fish` the storage temperature (e.g., tuna 35 degrees Fahrenheit, carp 40 degrees Fahrenheit). Additionally, a bulk resource (resource type) contains two additional fixed properties: `QoHUnit` (Quantity on Hand Unit) as a string and `QoHDataType` (Quantity on Hand Data Type) as an integer. `QoHUnit` defines the unit of measurement (e.g., **pounds** as in `300 pounds of tuna`). `QoHDataType` defines the data type of the unit (e.g., data type **double** for **pounds**).

V. REA-DSL to Relational Model Mapping

In this section we create a model mapping between the REA-DSL and a relational model to automatically generate a relational model from the REA-DSL models in terms of SQL statements. Still, the mapping and appearance of the relational model could be fine tuned by flags and preferences set by the IT professional at a latter stage. The generated SQL statements are widely accepted and may be imported by many database systems and database modeling tools. In our case we used the free MySQL Workbench [14] to import and show the relational model diagrams.

For educational purposes, we believe that the mapping is best explained by using the concrete syntax of the REA-DSL and the relational model. The exact mapping is described by a pseudo-code.

General mapping rules. Agents and resources can have generalization relationships in REA. For mapping the *generalization hierarchies* of the REA-DSL there are several common approaches to a relational model in literature [9], [10], [11]. We decided to create a separate table for each super and sub class where the sub class table references the super class table. This allows us to either directly reference the super or sub class which is needed by the REA concept *events*.

For all REA concepts with properties (i.e. agent, resource, and event) the object properties (green compartments) and type properties (yellow compartments) of the REA concepts will always become columns of the corresponding tables. Additionally, properties marked as *isPrimaryKey* and depicted by a key symbol become the primary key column of the table.

Agent mapping. On the left side of Figure 3 an example of an *agent hierarchy* is depicted. This generalization is considered to be complete and disjoint which is the most common case in REA models. It shows the super agent Employee, which can either be a Cashier or a Driver. On the right side the corresponding relational model is shown with the mapping indicated by the dashed arrows annotated by numbers between the models. These numbers are also referred to in the following text in brackets. The exact agent mapping rules are described as pseudo code in Algorithm 1.

A primary key column is indicated by a ”P” in a circle and a foreign key column by an ”F” in a circle. In a first step, we need to create a table for the possible agent types. Thus, each *super agent* (`Employee`) is mapped (2) to an agent type table (`AT_EmployeeType`) with type property columns (`StartingWage`). Additionally, an agent type table gets a `TypeName` column which also becomes the primary key if no specific primary key was specified in the type properties. The sub agents of the generalization hierarchy on the left actually specify, which types of employees are allowed. Thus, the names of the sub agents get inserted into the agent type table, in our case the sub agents `Cashier` and `Driver`.

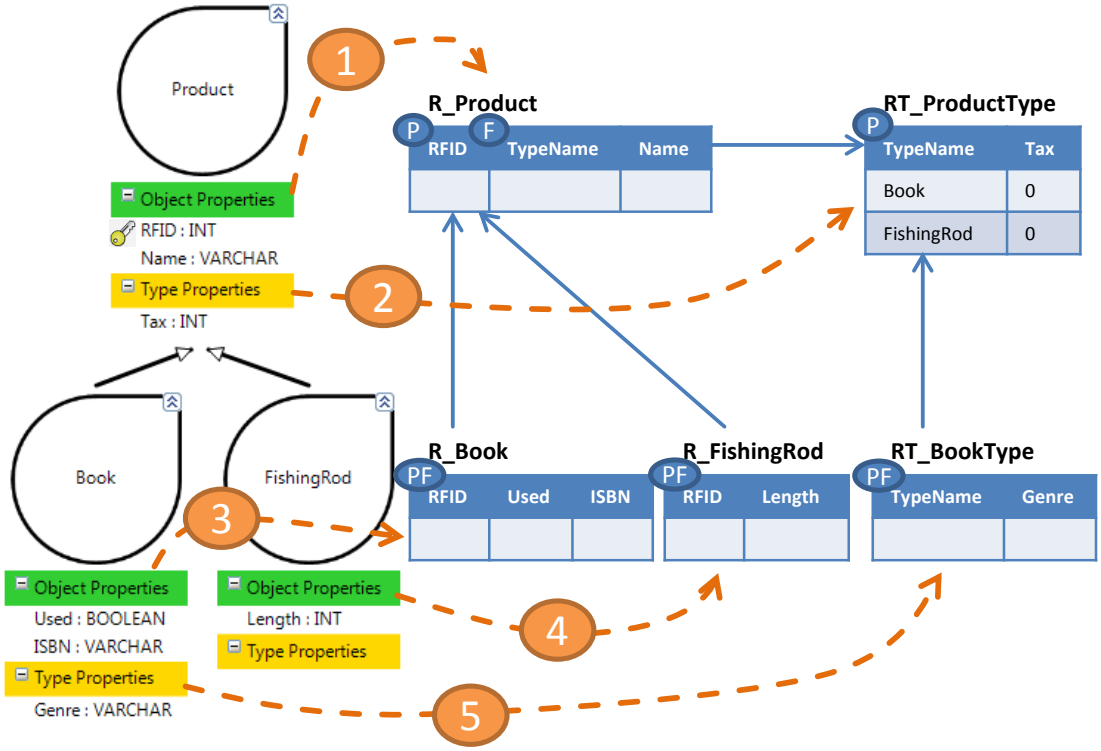

Fig. 4. Resource Mapping

Algorithm 1 Agent Mapping Rules

```
for all Super Agents do
    create Agent Type Table with the name <Agent.Name>Type;
      - add column TypeName of type VARCHAR and make it the primary key;
      - add all Type Properties as columns with the specified type;
      - insert the name of each related Sub Agent into the table;
    create Agent Table with the name <Agent.Name>;
      - add all Object Properties as columns with the specified type;
      - make the primary key object property the primary key column;
      - create foreign key to the Agent Type Table;
end for
for all Sub Agents do
    if Sub Agent has Type Properties then
        create Agent Type Table with the name <Agent.Name>Type;
          - add column TypeName of type VARCHAR and make it the primary key;
          - make column TypeName a foreign key to its Super Agent Type
        Table;
          - add all Type Properties as columns with the specified type;
    end if
    if Sub Agent has Object Properties or generate all sub agent tables is true then
        create Agent Table with the name <Agent.Name>;
          - add all Object Properties as columns with the specified type;
          - create same primary key as in Super Agent and make it a foreign key to
        it;
    end if
end for
```

In a second step, we create for each *super agent* an agent table (1) which will hold all individual agents. In our example we create an `A_Employee` table with all the object properties `Name`, `Age`, and `EmployeeId` (primary key) as columns. An additional column `TypeName` references the agent type table `AT_EmployeeType`. Thus, an `A_Employee` row entry can either be an `AT_EmployeeType` `Cashier` or `Driver`.

Each of the sub agents (`Cashier` and `Driver`) become their own agent table (`A_Cashier` and `A_Driver`) with the object properties columns (`LicenseNr` for `A_Driver`) (3,5) regardles of having specific properties or not. Consequently, each row entry for `A_Cashier` or `A_Driver` also has to have a row entry in `A_Employee`. They reference their super agent (`Employee`) with the primary key `EmployeeId`. If a sub agent (`Driver`) also has type properties, a specific agent type table (`AT_DriverType`) referencing the super agent table (`AT_EmployeeType`) is created (4). Note, there is no reference between `A_Driver` and `AT_DriverType` since it is implicitly referenced through `A_Employee` and `AT_EmployeeType`.

Resource mapping. The resource mapping (cf. Figure 4) is similar to the agent mapping and thus, is just briefly explained here. Due to space limitations we do not show the resource mapping pseudo code here but refer to our web site http://umm-dev.org/rea-mapping/.

The super resource (`Product`) becomes (2) a resource type table (`RT_ProductType`) with the type property column `Tax` and the primary key column `TypeName`. The sub resources `Book` and `FishingRod` become (3,4) row entries. Furthermore `Product` also becomes (1) a resource table `R_Product` with the object property columns `RFID` and `Name` as well as a `TypeName` column referencing the resource type table `RT_ProductType`. Each sub resource (`Book` and `FishingRod`) also become a separate agent table (`R_Book` with columns `Used` and `ISBN`, and `R_FishingRod` with column `Length`). If a sub resource

has type properties (Book) (5), a separate resource type table (RT_BookType with column Genre) referencing the super resource type table (RT_ProductType) is generated.

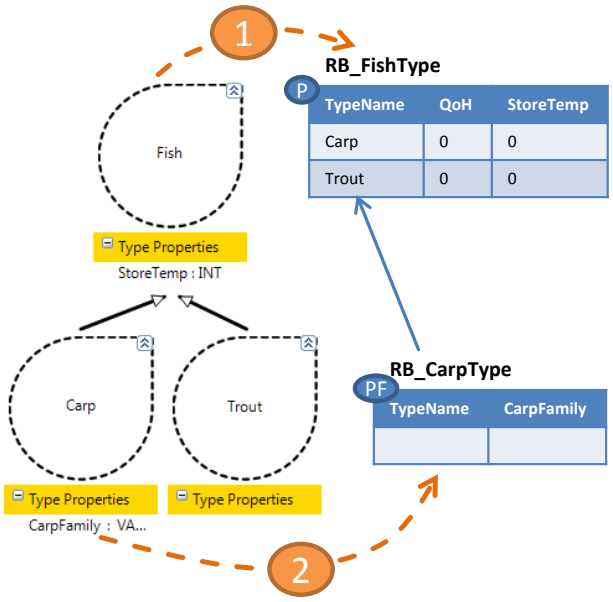

Fig. 5. Bulk Resource Mapping

As mentioned earlier there are special resources called bulk resources only containing type properties. According to the mapping in Figure 5, the super bulk resource Fish is mapped to the bulk resource type table RB_FishType (1) including the type properties as columns (StoreTemp) and the primary key column TypeName. Additionally, a quantity on hand (QoH) column for storing the amount of the bulk resource is created. All the sub bulk resources get inserted as row entries into the super bulk resource table (RB_FishType). If a sub bulk resource has specific type properties (e.g. CarpFimily in Carp), a separate bulk resource table (RB_CarpType (2) with CarpFamily column) is created referencing the super bulk resource table (RB_FishType).

Duality mapping. So far we concentrated on the mapping of the REA building blocks agents and resources. We already learned that these building blocks are associated with events in the duality model. In the duality mapping (cf. Figure 6 and Algorithm 2) we focus on the mapping of events that are in duality relationship according to the ”give and take” principle.

A duality model exists for each value activity of the value chain. As an example for the mapping we take the Selling duality as depicted on the right side of Figure 1. Figure 6 shows the mapping of the Selling duality (1) to a duality table D_SellingDualityTransfer with the primary key DualityId. Each event of the duality is mapped to an event table. For the Sale event (2) an event table E_SaleEvent is created with all the event properties (SaleNr and SaleDate) as columns and the SaleNr as primary key. Furthermore the SellingDualityTransferId column references the duality it belongs to and SaleEventTypeName

Algorithm 2 Duality Mapping Rules

```
1: create Duality Table with the name
   <PlanningModel.Name>DualityTransfer/Transformation
2:   - table name either Tranfer or Transformation according to model type
3:   - add DualityId as primary key column
4:   - add foreign key ContractId to the ContractTable
5: for all Events do
6:     create an Event Table
7:       - add all Object Properties as columns with the specified type
8:       - make the primary key object property the primary key column
9:       - add foreign key to the Duality Table and the Event Type Table
10:      - add foreign key to the Commitment Table which this event fulfills
11:    for all Connected Agents (participations) do
12:       if Single Agent then
13:          foreign key to Agent Table and add minutes used if is inside agent
14:       else if Multiple Agent then
15:          create a EventParticipateAgents Table
16:            - foreign key to Event Table and Agent Table
17:            - add minutes used column if is inside agent
18:       end if
19:    end for
20:    for all Connected Resources (stock-flows) do
21:       if Single Resource then
22:          foreign key to Resource Table and add minutes used if isUsed
23:       else if Multiple Resource then
24:          create a EventStockFlowResources Table
25:            - foreign key to Event Table and Resource Table
26:            - add minutes used column if isUsed
27:       else if Single Bulk Resource then
28:          foreign key to Resource Type Table and add minutes used if
             isUsed
29:          add quantity (of quantity type) column
30:       else if Multiple Bulk Resources then
31:          create a EventStockFlowResourceTypes Table
32:            - add Id as primary key column
33:            - foreign key to Event Table and Resource Type Table
34:            - add quantity (of quantity type) column
35:            - add minutes used column if isUsed
36:       end if
37:    end for
38: end for
```

references the event type of the Sale event (generated in the planning mapping).

Single participating agents (Salesman and Customer) are referenced directly from the event table (SalesmanEmployeeId and CustomerCustomerId). Additionally, for inside agents (Salesman) we also record the time spent for this event in a separate column (SalesmanMinutesUsed). Multiple participating agents (ShopAssistant) become (5) their own participation table E-A_SaleEventParticipateShopAssistant. The table has columns for referencing the SaleEvent and the ShopAssistant as well as the MinutesUsed. Multiple resources (Product) become (4) their own stock-flow table (E-R_SaleEventStockFlowProduct) referencing the corresponding SaleEvent and the individual Product. Multiple bulk resources Fish also become (3) their own stock-flow table (E-RB_SaleEventStockFlowFishType) referencing the corresponding SaleEvent and the FishType. Additionally to resources, for bulk resources a quantity column is created (FishTypeQuantity). Notice, resources can be marked as being used instead of being exchanged. In such a case, also an additional column ”‘minutes used”’ is generated.

The Payment event is mapped (6) to an event table (E_PaymentEvent) in a similar way as the Sale

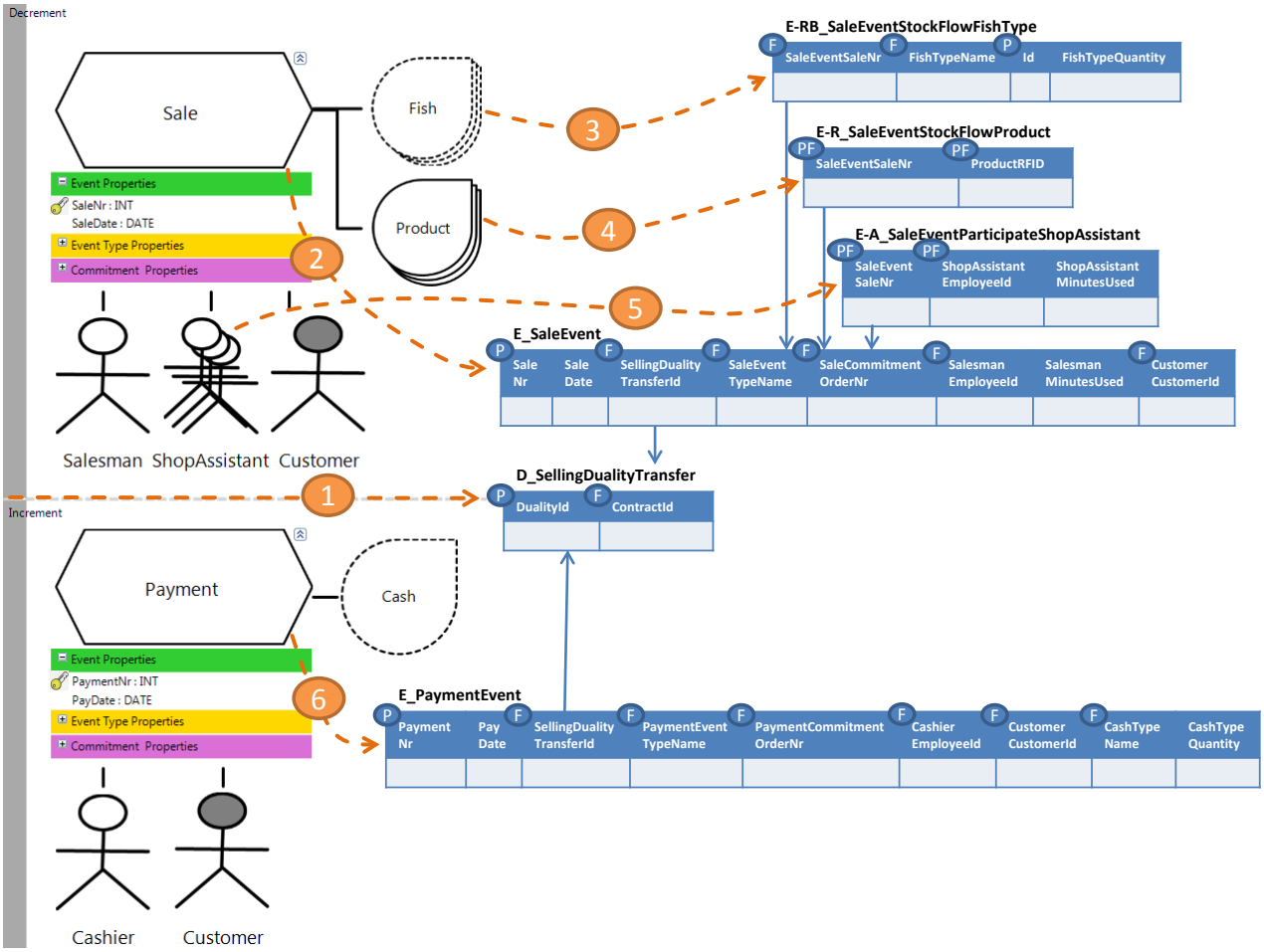

Fig. 6. Duality Mapping

event. As an additional concept, the single bulk resource `Cash` is directly added as column `CashTypeName` in the payment event table referencing the `CashType` and a `CashTypeQuantity` column storing the amount of `Cash` being exchanged.

VI. Mapping Evaluation

For the creation of our REA-DSL and the mapping to the relational model we followed the design science approach in information systems by Hevner [15]. Our designed artifacts are the REA-DSL consisting of a well-defined meta model and a graphical representation as well as the mapping rules for generating relational models. For designing the REA-DSL we specifically followed the methodological steps defined by Strembeck and Zdun [16]. Consequently, we started with (1) the identification of elements in the REA ontology, (2) derived the abstract syntax of the REA model in multiple revision cycles, and (3) defined the behavior of the REA-DSL and its elements. After we have reached a stable state, we defined (4) the concrete syntax of the REA DSL and (5) implemented a tool supporting the REA-DSL modeling. As a last step (6) we automatically generate an IT artifact – i.e. the SQL statements – by applying the mapping rules introduced in this paper on the REA-DSL model.

Our evaluation is threefold: (i) the technical feasibility is demonstrated by a tool implementation, (ii) a functional test based on existing REA models is conducted, and (iii) expert interviews are taken. In previous work [5], [6] we created the REA-DSL designer tool which allows to model REA-DSL models. In a first step we extended the tool with property compartments and primary key tags. In a second step we created T4 text templates [17] – a text template transformation language introduced by Microsoft – which implement the mapping rules presented in this paper. These T4 text templates take the REA-DSL models as an input and automatically generate relational models by the means of SQL statements (compatible with MySQL). We took these SQL statements and loaded them into the MySQL Workbench to get a graphical representation of the relational model. An excerpt of the T4 text templates for agents (1), its SQL statements output (2) and a relational graphical representation of the MySQL Workbench (3) is shown in Figure 7. After applying the complete mapping on the whole example presented in this paper, 73 tables conforming to the REA ontology were generated.

As for the functional test, we took 32 REA-DSL examples

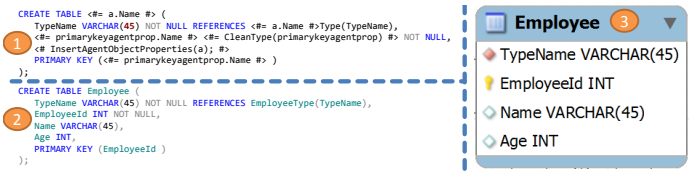

Fig. 7. T4 Template to SQL Statements to Relational Model

created in [6], extended them by properties and primary keys, and run the automatic T4 text template mapping on them. This resulted in 32 relational models which we successfully proved valid against the REA ontology and its axioms.

Finally, we consulted various experts on our results. In weekly conference calls with the founder of REA – William McCarthy – we received valuable feedback which we applied on the REA-DSL as well as on the mapping to the relational model. Furthermore, we showed three REA experts and twelve experts in conceptual modeling our REA-DSL models and the generated relational models. Afterwards, we asked them for the comprehensibility of the REA-DSL models compared to the REA class-like representation and to check the relational models for completeness and semantic correctness. All experts mentioned that the REA-DSL models are more intuitive and easier to understand and proved the derived relational models as correct.

VII. Summary and Future Work

When it comes to modeling accounting information systems (AIS) the Resource-Event-Agent ontology (REA) is a well-accepted business modeling ontology. REA allows to model the economic phenomena of a company within and outside its borders on an abstract level. These phenomenas are specified in REA by means of events exchanging resources between agents. Thus, it is able to capture the economic activities of the past and the present (e.g. a specific sale). However, the REA ontology had no dedicated graphical representation and therefore, was imprecise and hard to understand for business experts. We overcame this by developing a domain specific language for the REA ontology called the REA-DSL. This language can be used by business experts and IT professionals likewise to jointly create a conceptual model for an AIS in the requirement phase.

Unfortunately, these models cannot be processed and read by IT systems in order to automatically create database structures. This leads to an error prone and time intensive task for the IT professional of remodeling the REA-DSL to a relational model, which can be understood by database systems. Consequently, in this paper we present the extension of the REA-DSL by database concepts (i.e. properties and primary keys) and provide mapping rules to a relational model. We incorporated the new concepts and an automatic relational model generator based on the mapping rules into our REA-DSL tool. The tool now supports a thorough and easy creation of a relational database schema for AIS starting with the modeling of a conceptual REA-DSL model. We argue, that this fastens and streamlines the design of an AIS and at the end saves money in the development phase.

In this paper, we concentrated on the mapping of the operational layer of the REA-DSL [5]. In our paper [6], we have extended the REA-DSL to capture the commitments of the planning layer. Evidently, the planning layer has to be transformed to a relational structure as well. Due to space limitations, this mapping was out of scope for this paper and will be described in future work. Furthermore, we want to add configurations for the mappings in order to tune the relational model creation (e.g. choosing the mapping strategy for generalization hierarchies).

References

[1] M. Romney and P. Steinbart, *Accounting Information Systems*. Pearson Education, Limited, 2011.

[2] W. E. McCarthy, "The REA Accounting Model: A Generalized Framework for Accounting Systems in a Shared Data Environment," *The Accounting Review*, vol. 57, no. 3, 1982.

[3] G. L. Geerts and W. E. McCarthy, "An Ontological Analysis of the Economic Primitives of the Extended-REA Enterprise Information Architecture," *International Journal of Accounting Information Systems*, vol. 3, no. 1, pp. 1 – 16, 2002. [Online]. Available: http://www.sciencedirect.com/science/article/B6W6B-45BV92C-1/2/104bd15205cf9667808ffb157ba8fcd5

[4] J. Gordijn and H. Akkermans, "E3-Value: Designing and Evaluating e-Business Models," *IEEE Intelligent Systems*, vol. 16, no. 4, pp. 11–17, Jul–Aug 2001.

[5] C. Sonnenberg, C. Huemer, B. Hofreiter, D. Mayrhofer, and A. Braccini, "The REA-DSL: A Domain Specific Modeling Language for Business Models," in *Proceedings of the 23rd International Conference on Advanced Information Systems Engineering (CAiSE 2011)*. LNCS 6741: Springer, 2011, pp. 252–266.

[6] D. Mayrhofer and C. Huemer, "Extending the REA-DSL by the Planning Layer of the REA Ontology," in *Proceedings of the 7th International Workshop on Business/IT-Alignment and Interoperability (BUSITAL 2012)*. Springer, 2012, submitted to.

[7] G. L. Geerts and W. E. McCarthy, "The Ontological Foundations of REA Enterprise Systems, Tulane," March 2005. [Online]. Available: http://www.msu.edu/user/mccarth4/tulane.doc

[8] G. L. Geerts, , and W. E. McCarthy, "Policy-Level Specification in REA Enterprise Information Systems," *Journal of Information Systems*, vol. 20, no. 2, pp. 37–63, 2006.

[9] C. J. Date, *An Introduction to Database Systems (8. ed.)*. Addison-Wesley-Longman, 2003.

[10] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems, 6th Edition*. Addison-Wesley-Longman, 2010.

[11] L. Cabibbo and A. Carosi, "Managing Inheritance Hierarchies in Object/Relational Mapping Tools," in *Proceedings of the 17th International Conference on Advanced Information Systems Engineering (CAiSE 2005)*, 2005, pp. 135–150.

[12] G. Poels, "Conceptual Modeling of Accounting Information Systems: A Comparative Study of REA and ER Diagrams," in *Workshop on Conceptual Modeling Quality, Lecture Notes on Computer Science (ER 2003)*, 2003, pp. 152–164.

[13] G. L. Geerts and W. E. McCarthy, "Modeling Business Enterprises as Value-Added ProcessHierarchies with Resource-Event-Agent Object Templates," in *In Business Object Design and Implementation*. Springer-Verlag, 1997, pp. 94–113.

[14] *MySQL Workbench 5.2*, Oracle, 2011, http://www.mysql.com/products/workbench.

[15] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design Science in Information Systems Research," *MIS Quarterly*, vol. 28, no. 1, pp. 75–105, 2004.

[16] M. Strembeck and U. Zdun, "An Approach for the Systematic Development of Domain-Specific Languages," *Softw., Pract. Exper.*, vol. 39, no. 15, pp. 1253–1292, 2009.

[17] *T4 Text Template Transformation Toolkit*, Microsoft, 2011, http://msdn.microsoft.com/en-us/library/bb126445.aspx.