Extending the REA-DSL by the Planning Layer of the REA Ontology

Dieter Mayrhofer and Christian Huemer

Business Informatics Group, Vienna University of Technology, Austria,
{mayrhofer,huemer}@big.tuwien.ac.at

Abstract. The Resource-Event-Agent (REA) ontology is a powerful and well accepted approach towards the design of accounting information systems (AIS). However, the REA notation - that is currently based on class diagrams - is not very intuitive for business experts. Accordingly, we aim at a REA domain specific modeling language that facilitates the communication between business experts and IT professionals. In previous work we defined the REA-DSL operational layer reflecting actual business events which "have occurred" or "are occurring". In this paper we extend the REA-DSL by the planning layer capturing what future events "are scheduled" or "are planned" by commitments. Now, our REA-DSL covers all basic concepts to describe a full accounting infrastructure. The REA-DSL may serve as a solid basis for generating a conceptual AIS data model - which is subject to future work.

Key words: REA, business models, domain-specific language

1 Introduction

According to Romney and Steinbart an accounting information system (AIS) is a system that collects, records, stores, and processes financial and accounting data to produce information for decision makers [1]. They define the following main functions of an AIS: (i) collect and store data about events, resources, and agents; (ii) transform that data into information that management can use to make decisions about events, resources, and agents; and (iii) provide adequate controls to ensure that the entity's resources including data are available when needed as well as accurate and reliable. Accordingly, financial and accounting related data is analyzed, prepared, calculated, and visualized in order to provide insight into a company's current financial status as well as of its historic activities. Furthermore, the data helps to predict the financial future of a company and, thereby, helps managers in their decision making.

In order to create a useful and meaningful AIS, the data structure and user interface has to reflect the economic phenomena on which companies base their business. Thus, it is essential that the business people providing the requirements can unambiguously communicate with the IT professionals which are in charge of creating the AIS. Business ontologies - providing abstract descriptions of enter-

prises in their business context - may be used as a language for communicating these requirements between domain experts and IT staff.

The most prominent business ontology for accounting information systems is the Resource-Event-Agent (REA) ontology developed by McCarthy, Geerts and others [2]. REA is a widely accepted framework for the design of a conceptual model of the accountability infrastructure of enterprise information systems. Originally, REA targeted the resource flows within and between companies describing what is currently occurring and what has occurred in the past. This is known as the operational layer. Later it was extended by a planning layer and a policy layer capturing what should, could, or must be occurring sometime in the future [3, 4].

Today, REA may be considered as a powerful business ontology capturing all relevant data to generate the conceptual design of an AIS. However, we feel that it does not deliver an appropriate representation of the business model which can be understood not only by the IT expert, but also by the business expert. Thus, the use of REA in the design of AIS does not yet reach its full potential. We argue, that an easy-to-understand REA notation will accelerate, streamline, and reduce the costs of the AIS development process.

Consequently, we started the endeavor of developing a domain specific modeling language for the REA concepts which aims at both (i) delivering an intuitive REA notation and (ii) retaining the full expressiveness of the REA concepts. Similarly to the development of REA itself, our REA-DSL started on the operational layer only [5]. In this paper, we are going to extend the REA-DSL by concepts of the planning layer and types of the policy layer. Once we have consider all REA layers, we are able to derive a conceptual data model, e.g. an entity-relationship diagram, for the resulting AIS. However, this transformation process is out of scope for this paper and will be addressed in future work.

The remainder of this paper is structured as follows: In Section 2 we give a brief introduction of the REA ontology with focus on the commitment and type artifacts. An example of the REA-DSL is given in Section 3. Section 4 defines the extended REA-DSL. The evaluation is provided by our tool in Section 5 and we conclude the paper by the summary in Section 6.

2 Related work on REA

When developing a domain specific modeling language for REA, related work focuses more or less only on papers related to REA. A comparison with alternative approach for a domain specific modeling language is impossible, since best to our knowledge no such efforts have been published. Accordingly, we concentrate in the following on REA and in particular on the REA planning layer and types of the policy layer.

REA was originally proposed as a reference framework to conceptualize the resource flows within and between firms in terms of what is currently occurring or what has occurred in the past [2]. Accordingly, REA focuses on the economic exchanges as the central unit of analysis. Instead of representing these exchanges

with double-entry bookkeeping artifacts (e.g. debits, credits, accounts), REA proposes concepts and patterns to derive semantic models of economic exchanges and transformations [6].

Following its name, REA is based on the three main concepts: economic resources, economic events, and economic agents. It should be noted that for a better readability we drop the prefix term *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 30 November 2011 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 when `Fred` (*agent*) pays the amount of `700 Euros` to the `cashier Mark` (*agent*).

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` using the concepts of resource, events, agents, duality, stockflow, and participation. The relationships between these concepts are formalized on the M2 layer as a meta model which we already published at CAiSE 2011 [5].

The above mentioned example demonstrates that the original REA [2, 7, 6] is capable to record events that have already happened. As mentioned earlier, AIS should provide management an insight into a company's current financial status in order to make proper decisions. Usually, the financial status depends not only on the past, but also on commitments that the company has already made for the (near) future. Thus, it is desirable to extend REA by such commitments and internal plans to fulfill these commitments. Accordingly, REA should be able to capture the following example of a reservation: On 2 November 2011 the `salesman Joe` *commits* himself - as a particular `salesman` - and two (not yet known by name) `shop assistants` to give `customer Fred` at least `40 pounds of tuna`. In return, `customer Fred` *commits* himself to pay a (not yet known by name) `cashier`. It is easy to recognize, that the above mentioned actualization of the events is in-line with the commitments, even if the actualization overfulfills the commitment by additional `10 pounds of tuna` and the `fishing rod`.

In order to accommodate such a scenario REA has been extended by a planning and policy layer [3, 8, 4]. These layers cover economic activities that should, could, or must happen in a company and are, thus, relevant for planning and controlling activities. The extension allows to model the previous commitment example on an abstract level on M1. Therefore, new concepts are introduced on the meta model layer M2. The most central concept is a legal *commitment*. The concept *commits* associates the *agent* who does the *commitment* to the *commitment*. *Reciprocity* is a relationship between *commitments*. The concept of *fulfill* links *commitments* and associated *events*. *Reserve* is used to associate *resources* or *agents* to *commitments*. Furthermore, REA introduces the *typification* to sup-

port the planning and policy layer. From the above example it is evident that in some cases one may already refer to a particular, yet-known *agent* (such as the `salesman Joe`), and in other cases one may only refer to the *type of agent* (such as `shop assistant`), because it is not-yet-known in person. This is reflected by the M2 concept of *agent type*. Corresponding concepts exist for *event types* and *resource types*. An overview of the REA M2 meta level concepts is presented in Figure 1.

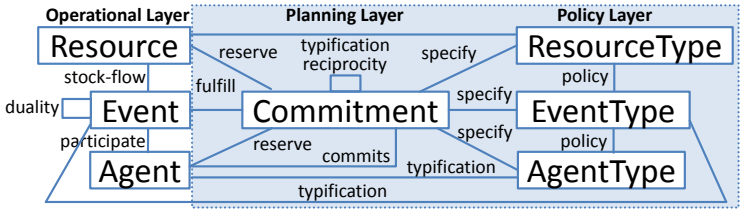

Fig. 1. REA Meta Model

Accordingly, we aim at supporting all these concepts in our REA-DSL. Similarly as the papers on the planning and policy layer [3, 8, 4] extend the basic REA concepts [2, 7, 6], this paper extends our first paper on the basic REA concepts [5]. In addition, our work in this paper was influenced by Gailly et al. [9]. They already tried to formalize types and commitments but did not provide a domain specific language and separate views for it.

3 REA-DSL Example

Before going into all the theoretical details when explaining the meta model of our REA-DSL, we start illustrating our results by a simple, but still realistic example. The example in this paper is based on *Sy's Fish* - an example that was used also by Geerts at al. [7] to demonstrate REA.

Example business model. *Sy's Fish* purchases fishes from the fish market. He sells three different kind of fishes: carp, trout, tuna. Additionally, he acquires products from a factory: books and fishing rods. The fishes and products are transported to *Sy's Fish* by leased trucks. At *Sy's Fish* all the fishes get cleaned. The products and fish are then sold to customers in order to make profit. To be able to accomplish all tasks, *Sy's Fish* employs a couple of employees, which can either be a salesman, shop assistant, or cashier.

Modeling REA. The REA-DSL consists of five different interlinked views: (i) *agents view*, (ii) *resources view*, (iii) *value chain view*, (iv) *planning view*, and (v) *operational view*. You might start modeling by using any of the views. In the following, we provide an example of our REA-DSL.

Agents view. We start modeling the business model with the REA-DSL by defining the different agents. In the agents view (cf. Figure 2a) the agents inside the company are depicted on the left side with the white head stick figures. The general agent is an `employee`, which can either be a `salesman`, a `shop assistant`, or a `cashier`. The outside agents are depicted as black head stick figures. *Sy's Fish* does business with the outside agent `customer`. For the moment

we forget about other outside agents which are involved in the overall process, but are not relevant for understanding the example.

Resources View. Next, we define all resources which need to be tracked and recorded in *Sy's Fish* company. These resources are depicted by the shape of a drop in Figure 2b. Resources which can be identified individually are marked by a solid drop, whereas bulk resources where the individual real world objects cannot (or need not to) be identified are marked by dashed drops. Accordingly, the resource `product` is a solid drop. Products can be categorized into `books` and `fishing rods`. Another resource is `cash`. `Cash` is a bulk resource that appears as a dashed drop. Evidently, *Sy's Fish* cannot track each single coin or bill and therefore, only the whole amount of `cash` is of interest. Another bulk resource is `fish`. `Fish` can be categorized into `carp` and `trout`.The last resource is `truck`, which is modeled as a solid drop.

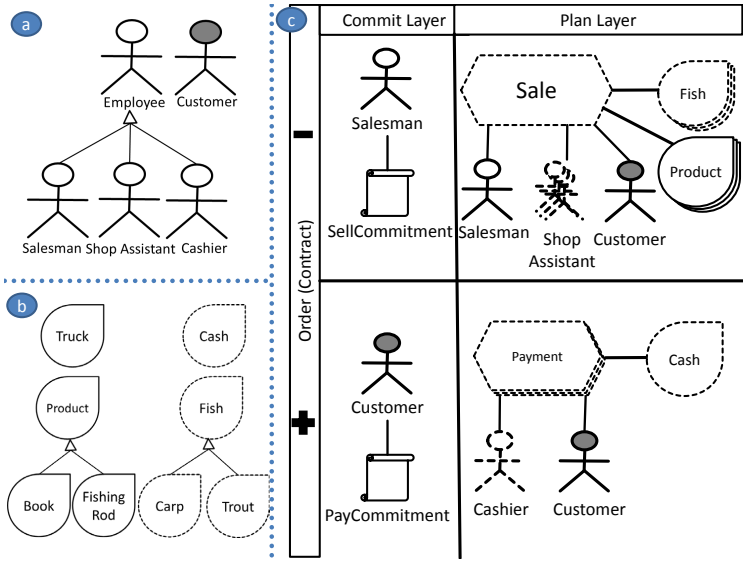

Fig. 2. (a) Agent View, (b) Resource View, and (c) Planning View

Value chain view. In the next step a high level overview of *Sy's Fish* business model is provided by depicting it in a value chain view (cf. Figure 3). The value chain contains economic activities that create higher value by value transfers with external partners or transformations inside the company. Usually, resources (which are referenced in the resource view) created by one value activity serve as input to another one. In order to keep Sy's Fish value chain as simple as possible we omit to depict the resource labor which is naturally input to all economic activities.

`Fish` and `products` are purchased with `cash` in the two value activities `fish buying` and `product buying`. The `products` and `fish` are `transported` to *Sy's Fish* by a `truck` leased for `cash` in the `truck acquisition` value activity. The `fish` is cleaned in the `cleaning` value activity and then the `products` and `fish` are sold for `cash` in the `selling` value activity.

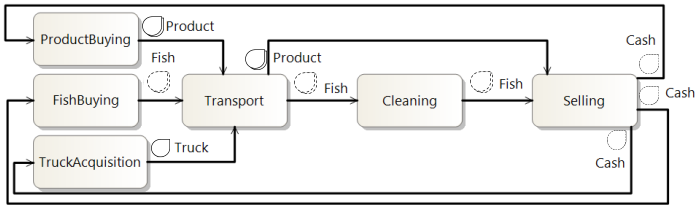

Fig. 3. Value Chain View

Planning view. Each value activity defined in the value chain view can be refined in a referenced planning view. For this example we just elaborate on the `selling` value activity which is depicted in Figure 2c. It also references agents and resources defined in the agents and resources view. The resulting contract is a 2 x 2 matrix. The top covers a commitment leading to a decrease in resources, and the bottom defines the compensating commitment leading to an increase in resources. Orthogonally, the left hand side defines who does the commitment and the right hand side defines what it is about and who is going to fulfill it.

The contract is an `order` and contains two commitments: the `salesman` committing in the `sell` commitment to engage in the `sale` event in the future and the `customer` committing in the `pay` commitment to engage in the `payment` events in the future. A concrete `salesman` will participate in the future `sale` of the resources `fish` and `product`. Since more than one `fish` and more than one `product` are potentially sold, their shapes appear as *stacks of drops*. As mentioned before `fish` are bulk resources with dashed drops and `products` are individually identifiable with *solid drops*. Note, a solid drop in the commitment means also that the individual `product` being part of the future `sale` is exactly defined at the time of the commitment. If only the type of the `product` is specified in the commitment it has to appear a *dashed drop* (or *stack of dashed drops* in case of multiple `products`). Furthermore, the `salesman` will be supported by one or more `shop assistants` (depicted as a stack of dashed stick figures) which are not known at the time of the commitment. These `shop assistants` will get concrete when the `sale` event happens. In the reciprocal commitment, the `customer` commits himself to pay in `cash` to a not-yet-known `cashier`. The staple of `payment` events signifies that there is the option of multiple (partial) `payments`.

In this example we do not further elaborate on the operational view which was the main focus of our CAiSE 2011 paper [5]. The interested reader may refer to this paper. However, it should be noted that one may semi-automatically transfer the right hand side of the contracts in the planning view to the operational view.

4 REA-DSL Formalization

Having introduced Sy's Fish example, we are now elaborating on the formal concepts which this example is based upon. These concepts are represented in the REA-DSL meta model on the M2 layer. We concentrate on the DSL meta

model of the planning view. Due to space limitations we do not describe the meta model of the agent view as well as the the resource view, the operational view, and value chain view, which we slightly extended for the incorporation of types but besides that already covered in [5].

In [5] we have described *exchanges* including *events* in the *operational view*. These exchanges often do not happen unexpected. There can be *commitments* to fulfill events in the future. Such commitments can for example be an `order` to `buy fish` or a schedule to `clean fish`. Thus, events happening in the future are planned beforehand. The major contribution of this paper is to add this commitment concept together with types to the REA-DSL by providing an additional view called the *planning view*. This view makes use of resources and agents defined in the resource and agent view. Following we propose the *planning view* meta model (cf. Figure 4) and its concrete syntax (cf. Figure 5). Classes in the meta model which also have a corresponding stencil in the concrete syntax of the abstract model are marked with a numbered circle.

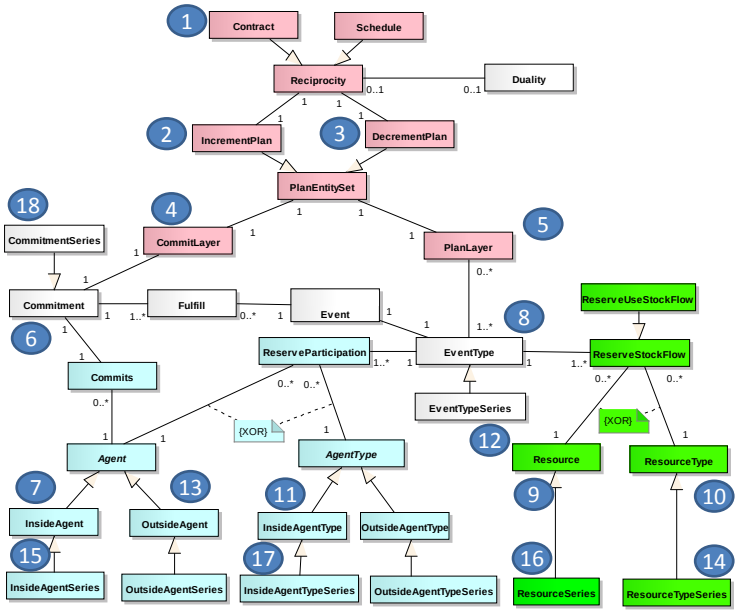

Fig. 4. Commitment Meta Model

Reciprocity. The root element of the planning view is the *reciprocity*. A reciprocity is analog to the *duality* in an operational view. The duality connects the incrementing entities with the decrementing entities. Likewise, reciprocity connects the *incrementing commitments* with the *decrementing commitments* in the planning view. The reciprocity may be associated with a duality which refers to the corresponding duality in the operational view and vice versa. Thus, it is the link between the planning view which plans future events and the operational view which shows the events at the time of execution. Notice, if the reciprocity is conceptually congruent to the duality, the duality does not have to be modeled separately.

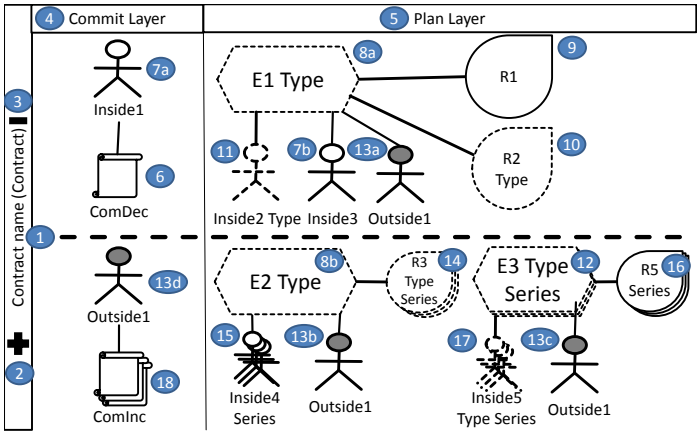

Fig. 5. Commitment Abstract Example

Reciprocity can either be a *contract* (1) or a *schedule*. The abstract example shows a contract (1) (indicated by `contract` in the brackets) with the name `contract name`. In the case of a schedule the term in the brackets would be `schedule`. If the reciprocity is a contract the corresponding duality in the operational view has to be a transfer and if it is a schedule the corresponding duality has to be a transformation, respectively.

Increment plan and decrement plan. The reciprocity consists of two *plan entity sets* called *increment plan* (2) and *decrement plan* (3). The increment plan is depicted as the lower swimlane annotated by a *plus sign* and contains all the commitments and related event types which will lead to an increment of resources in the future. Accordingly, the decrement plan is depicted as a swimlane annotated by a *minus sign* and contains all the commitments and event types which will lead to an decrement of resources in the future.

The *increment plan* (2) and *decrement plan* (3) are divided into two layers: the *commit layer* (4) on the left side and the *plan layer* (5) on the right side.

Commit layer. In general, the commit layer contains the *commitment* (6) which is made on future events defined by *event types* (8) in the plan layer. One *agent* legally *commits* to the *commitment* (6) which is depicted by a scroll. Agents can either be *inside agents* (7) or *outside agents* (13). A commitment on the decrement plan always has to be in reciprocity with a commitment on the increment plan in order to provide the rational of individual economic activities. In a *contract*, an *inside agent* (7) has to commit to the commitment in the decrement plan and an *outside agent* to the commitment in the increment plan. As for a *schedule*, only *inside agents* can commit to the *commitment*. These restrictions are defined by OCL constraints on the meta model. In the abstract example a commitment `comdec` (6) is made in the decrement plan (3) (upper lane) by the *inside agent* `inside 1` (7a) to execute *event types* (8) specified in the plan layer in the future. In return, a commitment `cominc` (18) is made by the *outside agent* `outside 1` (13d) in the increment plan (2) (lower lane) to

execute *event types* (8) specified in the plan layer in the future. The commitment `cominc` (18) is actually a *commitment series* depicted by a stack of scrolls (18). The special meaning behind a commitment series is, that many commitments (e.g., two orders) can be fulfilled by one event (e.g., a joint delivery).

Plan layer. The *plan layer* (5) on the right side in general specifies the type of future *events*, their *agents*, and *resources* being involved. These future *events* are defined in the corresponding *duality* of the operational view.

Event types. The plan layer can contain one-to-many *event types* (8) depicted as dashed hexagons. Contrary to the *operational view* where *events* are specified, in the planning view only *event types* are specified (e.g. regular sale type). The reason for this is, that at the time of planning the events, the actual future events cannot be referred to, because they simply do not exist yet. Thus, only the event type can be referred to. Instead of event types also the sub type *event type series* (12) depicted by a dashed stack of hexagons might be specified in a plan layer. An event type series specifies one-to-many event types of the same kind (e.g. a payment being split up in many partial payments). The event type is related to the *event* in the duality which *fulfills* a commitment (6) or commitment series (18) in the future. A commitment can be fulfilled by one-to-many events and one event can fulfill zero-to-many commitments. This means, that for a commitment, there must be at least one related event in the operational view, but for an event, there does not necessarily exist a commitment. In the abstract example `E1 type` (8a) and `E2 type` (8b) are regular event types and `E3 Type Series` (12) is an event type series. Accordingly, an event `E1` and `E2` as well as an event series `E3 Series` have to exist in the operational view.

Resources and resource types. *Event types* are connected to at least one *resource* (9) or *resource type* (10) by *reserve stock-flows*. These resources/resource types refer to resources/resource types in the resource view. If the event type resides on the increment plan (2) the resource will be gained in the future. Otherwise, on the decrement plan (3) the resource will be used in the future when connected by a *reserve use stock-flow* and consumed when connected by a regular *reserve stock-flow*.

A *resource* (9) connected to the event type (also called being reserved) specifies, that the exact identifiable resource is already known at the time of the commitment (e.g. a product with a specific RFID code). Consequently, the exact same resource is also referred to in the future event of the operational view. It is depicted by a regular *solid drop* (cf. 9, `R1`) and references an identifiable resource in the resource view. Similar, a *resource series* (16) has the same meaning for many resources (e.g., 3 books) and is depicted by a *stack of solid drops* (cf. 16, `R5 Series`). It also references an identifiable resource in the resource view.

A *resource type* (10) connected to the event type (also called being specified) can have two specific meanings. Either the referenced resource in the *resource view* is an (i) *identifiable resource* or it is a (ii) *bulk resource*. In the case of an identifiable resource (i), in the future event of the duality a specific resource of this type will be associated. However, at the time of the commitment the exact identifiable resource is not-yet-known (e.g. booking a double bed room in a hotel, but the specific room with number 704 will be assigned at the time of

arrival). This resource is depicted by a *dashed drop* (10, `R2 Type`). Similar, a *resource type series* (14) has the same meaning for many resource types (e.g., 10 tons of tuna and 12 tons of carp) and is depicted by a *stack of dashed drops* (14, `R3 Type Series`). In the case of a referenced bulk resource (ii) the actual resource can never be individually identified. Thus, it only defines the quantity of a resource type to be in a stock-flow of a future event (e.g. ordering 1000 liters of heating oil). Similar, a *resource type series* defines the quantity of many resource types to be in a stock-flow of a future event (e.g. ordering 1000 liters of heating oil and 70 liters of diesel).

Agents and agent types. *Agents (series)* and *agent types (series)* are reserved/specified for participation in *event types* (8) through *reserve participation* links. Reserved *agents* mean, that the individual agent can already be defined at the time of the commitment and are depicted by a *solid stick figure* (7b, `Inside 3`, 13, `Outside 1`). On the other hand, specified *agent types* mean, that only the type of agent can be defined at the time of the commitment and are depicted by a *dashed stick figure* (11, `Inside 2 Type`); once the commitment is fulfilled by a future event, this *agent type* becomes a concrete known *agent*. An example is specifying that two arbitrary `shop assistants` are needed in the future event, but we do not know the exact individuals yet. An *agent series*/*agent type series* always refers to one-to-many *agent*/*agent types* and is depicted as a *stack of stick figures* (15, `Inside 4 Series`, 17, `Inside 5 Type Series`).

Event types of *schedules* have at least one *inside agent* (7)/*inside agent type* (11) and no *outside agents*. On the other hand, *event types* of *contracts* (1) additionally can have *outside agents* (13). There is usually the same single *outside agent* used for every commitment and event type. However, in some rare cases there might be different *outside agents* as well as even *outside agent types*.

5 REA-DSL Evaluation

The development of REA has been following the *design science* approach in IS by Hevner [10]. In our work the designed artifact is the REA-DSL comprising a well-defined meta model and an appropriate graphical notation. For the design of the REA-DSL we followed the methodological steps for designing domain specific languages suggested by Strembeck and Zdun [11]: Accordingly, we started with (1) the identification of elements in the REA ontology. Next, we (2) derived the abstract syntax of the REA model including the core language model and the language model constraints and (3) defined the DSL behavior, i.e. determining how the language elements of the DSL interact to produce the intended behavior. Once we had reached a stable state, we defined the DSL concrete syntax (4). Finally, we implemented a modeling tool support for the DSL.

With respect to evaluation we (i) demonstrated the technical feasibility by means of a tool implementation, (ii) performed a functional test based on existing REA models, and (iii) conducted interviews with experts. The tool is based on the *Microsoft Visual Studio 2010 Visualization & Modeling SDK*. The REA-DSL tool incorporates the REA-DSL meta model as presented in this paper and in

our previous paper [5]. In addition, custom code enables to apply additional constraints to the REA-DSL models to adhere to the REA rules.

In the functional test we demonstrated that the REA-DSL is able to address all REA concepts in an appropriate and correct manner. For this purpose we took as input 32 example models which were modeled according to the class-like, original REA representation. Then, we modeled these 32 example models with the REA-DSL by means of our REA-DSL modeling tool. This task helped a lot in the refinement and adjustment of the REA modeling tool. Finally, we were able to describe all 32 models in the REA-DSL.

Eventually, we consulted different experts on our results. Most important we discussed our results with the founder of REA – William McCarthy – on weekly conference calls and incorporated his feedback. We showed them one and the same example in the class-like, original REA representation as well as in the REA-DSL. All the experts agreed that our REA-DSL notation is much more intuitive to describe the underlying business semantics and, thus, may better serve the communication between business experts and IT staff.

6 Summary and Future Work

The Resource-Event-Agent (REA) ontology is a well accepted approach for developing conceptual models for accounting information systems (AIS). The REA concepts are based on well established concepts of the literature in economic theory - which is certainly one of the strengths of REA. However, REA leaves space for diverging interpretations of the relationships between core concepts. Even worse, REA has no dedicated representation format and, thus, no graphical syntax. This is a major shortcoming given its goal to serve as a language for communicating requirements between domain experts and IT staff. Based on these shortcomings we started to develop a domain modeling language for REA that comes with both an unambiguous meta model definition and an intuitive graphical notation. Our first step concentrated on the REA operational layer [5]. This paper extends our previous work by adding concepts of the policy and planning layer to the REA-DSL.

In this paper we proposed extending the domain specific language REA-DSL by commitments and types of the REA policy and planning layers. REA has its roots in the accounting discipline and was extended to a business modeling ontology over the years. Before the REA-DSL was introduced by us in Sonnenberg et al. [5], the REA ontology lacked a dedicated representation built upon a precise formalization. These limitations are tackled by the REA-DSL. However, up to now the REA-DSL just incorporated the core REA concepts resource, event, and agent. The most significant new concepts are commitments, the typification of agents, resources and events, as well as the concept of bulk resources. Commitments define a legally binding obligation to engage in future events. Types can be seen as a grouping of objects with the same properties and can be referenced in commitments if the actual object is not known. Bulk resources refer to resources where the individual resource is not identifiable and traceable. All new

concepts have been added to our REA-DSL modeling tool which now covers the operational layer, the planning layer and types of the policy layer.

The ultimate goal of our approach is a business-driven database design or, in other words, the transformation of the REA-DSL artifacts to a conceptual data model for AIS. Accordingly, our future work will add the concepts of attributes and primary keys to resources, events, agents, commitments, etc. Once, this is accomplished transformation rules between the REA-DSL and Entity-Relationship-Diagrams or class diagrams have to be specified. We have already started to work on these issue and first results are very promising. Furthermore, we want to extend the REA-DSL by the concept of terms to model exceptions. Terms specify what happens, if a commitment is not fulfilled and therefore broken. Imagine ordering a fish and it is not delivered at the right time specified in the commitment. In this case terms apply and specify additional events such as penalty payments or additional provided rebates. Once we have completed the work on the conceptual data models and the concept of terms we want to conduct a real world case study by means of the REA-DSL tool.

References

1. Romney, M., Steinbart, P.: Accounting Information Systems. Pearson Education, Limited (2011)
2. McCarthy, W.E.: The REA Accounting Model: A Generalized Framework for Accounting Systems in a Shared Data Environment. The Accounting Review **57**(3) (1982)
3. Geerts, G.L., McCarthy, W.E.: An Ontological Analysis of the Economic Primitives of the Extended-REA Enterprise Information Architecture. International Journal of Accounting Information Systems **3**(1) (2002) 1 – 16
4. Geerts, G.L., , McCarthy, W.E.: Policy-Level Specification in REA Enterprise Information Systems. Journal of Information Systems **20**(2) (2006) 37–63
5. Sonnenberg, C., Huemer, C., Hofreiter, B., Mayrhofer, D., Braccini, A.: 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) 252–266
6. Geerts, G.L., McCarthy, W.E.: An Accounting Object Infrastructure for Knowledge-Based Enterprise Models. IEEE Intelligent Systems **14**(4) (1999) 89–94
7. Geerts, G.L., McCarthy, W.E.: Modeling Business Enterprises as Value-Added ProcessHierarchies with Resource-Event-Agent Object Templates. In: In Business Object Design and Implementation, Springer-Verlag (1997) 94–113
8. Geerts, G.L., McCarthy, W.E.: The Ontological Foundations of REA Enterprise Systems, Tulane (March 2005)
9. Gailly, F., Poels, G.: Towards Ontology-Driven Information Systems: Redesign and Formalization of the REA Ontology. In: Proceedings of the 10th International Conference on Business Information Systems. BIS'07 (2007) 245–259
10. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design Science in Information Systems Research. MIS Quarterly **28**(1) (2004) 75–105
11. Strembeck, M., Zdun, U.: An Approach for the Systematic Development of Domain-Specific Languages. Softw., Pract. Exper. **39**(15) (2009) 1253–1292