TOWARDS A GENERIC DATA MODEL FOR REA BASED APPLICATIONS

Bernhard Wally and Christian Huemer
Institute of Software Technology & Interactive Systems, Vienna University of Technology, Favoritenstr. 9-11, Vienna, Austria
wally@big.tuwien.ac.at, huemer@big.tuwien.ac.at

Keywords: Software Engineering, Business Models, Business Ontologies, Database Engineering.

Abstract: The original REA accounting model (McCarthy, 1982) has been extended in previous years into a business modeling language. Apart from its conceptual model, academic effort has been put into the definition of a formal description language, based on standards such as UML or OWL. The specification of a generic data model from a software engineering point of view for domain independent use of REA for business model specification and execution has been touched only briefly in the past. Thus, we present a data model concept for runtime-configurable REA business model definition and execution.

1 INTRODUCTION

The REA accounting model (McCarthy, 1982) was initially developed as a modern accounting methodology based on real world artefacts instead of the rather artificial double entry book keeping, originating in the Middle Ages. Over the years, it has been extended towards a business modeling language, which enables the description of value chains and enterprise resource planning (Geerts and McCarthy, 1997; Geerts and McCarthy, 2000; Gailly and Poels, 2007).

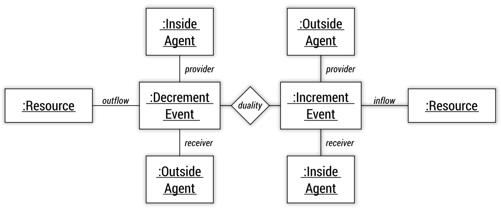

Figure 1: Sample object diagram of REA's core concepts.

Figure 1 depicts the REA core concept in terms of an UML object diagram: an economic exchange (*duality*) of goods comprises *increment* and *decrement events*, where the former represent events that add value to the company (e.g.receiving cash in a sales activity) and the latter are events that decrease the value of a company (e.g.handing over a product in a sales activity). Each of the events is related to a certain *economic resource* (cash and product in our example, respectively) and to two *economic agents*, the provider and the receiver of the economic resource.

The REA business model language has been used to model various application scenarios, however the data structure of these scenarios has been tailored to the specific needs of that scenario—it was not defined in a domain agnostic way. The approach found in the literature to implement REA applications is to declare concrete instances of entities of the REA meta model as classes in a programming language accompanied by corresponding database tables, see e.g. (Hrubỳ et al., 2006) for a "Pizza Delivery" and (Mayrhofer, 2012) for a "Fish Sale" sample application. In our approach, we are preparing a uniform infrastructure as a generic data model for the implementation of various application scenarios on top. We are thus designing the REA meta model as a multi-layered class hierarchy with corresponding database design and in turn enable specific applications to be modeled and executed based thereon.

Our effort on a generic data model for REA is in its beginnings—first concepts, findings and results have been published in (Gürth, 2014), (Mayrhofer et al., 2014) and (Wally et al., 2014). In this work we present details on the meta modeling methodology we are applying and its implementation concepts.

2 RELATED WORK

(Hrubý et al., 2006) is an extensive work on basic and advanced REA concepts and their realization in terms of software models. The presented workflow in the concrete examples follows an intuitive pattern: (i) the REA Meta model is modeled in terms of classes in an object oriented programming language, (ii) the domain specific model is again modeled and implemented in terms of software classes, and (iii) the domain model classes are instantiated at runtime for capturing and representing business artefacts.

In (Nakamura and Johnson, 1998) the Type Object pattern (Johnson and Woolf, 1997) is introduced to describe REA models—this notation is picked up also in (Hrubý et al., 2006; Geerts and McCarthy, 2006): REA concepts are described on a type and on an object layer, where instances on type layer define common properties for instances on the object layer. Also, this *typification* is used as a business model notation: in case a specific REA entity cannot be noted explicitly, its type object is used as a specification for what kind of entity would take part in a certain business pattern.

A model driven approach for the notation of REA based business models is presented in (Gailly and Poels, 2010), where the REA Meta model is defined as an UML profile, and the concrete domain model is modeled as a class diagram applying the REA UML profile. Again, this approach freezes the domain model at design time (expressed in UML) and requires different technology for the definition and the execution of business models.

In contrast to these approaches, we are proposing an integrated solution , where (i) the REA Meta model serves as language definition on MOF[1] layer M2 (modeled and implemented as concrete classes in an object oriented programming language), (ii) the domain model is dynamically modeled in M1 in form of instances of the previously mentioned classes, and (iii) the business artefacts on M0 are again instances of the model instances in M1. This approach allows the user of an enterprise resource planning system that is based upon REA concepts to declare additional exchange methods, resources or agents on-the-fly, and make use of them immediately. To foster M1–M0 interoperability, we strive for an approach that incorporates M1 and M0 models in a single storage and execution system. Interoperability with the approach described in (Gailly and Poels, 2010) could be realized via an import/export mechanism.

[1]Meta Object Facility (Object Management Group, Inc., 2013), see http://www.omg.org/spec/MOF/

2.1 Meta Modeling

In (Atkinson and Kühne, 2008) it is argued that two level modeling is introducing "accidental complexity" in cases where the domain scenario features three or more levels, because some workaround must be found to fit the many levels into the two levels provided. One such application scenario is when an element of a domain scenario requires to influence instances traditionally "out of reach" of that element (declare properties for instances of instances). For that purpose the concepts of "clabjects" (introduced in (Atkinson and Kühne, 2000)) and "deep instantiation/characterization" with "potencies" (Atkinson and Kühne, 2001; Kühne and Steimann, 2004) are introduced. It allows specifying properties that should be instantiated `potency` number of levels below its declaration. However, in our modeling approach, and for the given use case of REA business models, we agree with (Frank, 2011a), in that "potencies > 2 are not needed". In order to be able to model potencies of value 2, (Frank, 2011a) and (Frank, 2011b) introduce the concept of "intrinsic features/attributes/properties" (properties that are not implemented by instances but by instances of instances)—in a sense, intrinsic features are thus deeply instantiated properties with a hardcoded potency of 2. In (Yoder and Johnson, 2002) the "adaptive object-model" architectural style is described by discussing various concepts found in the literature and in implementations for the purpose of providing meta modeling support for the design and implementation of adaptive software systems. As a foundation for such systems the type object pattern and property pattern are identified, which together form a "type square".

In object oriented languages such as Java and C++, a single class declaration/definition is a sketch for four different data related concepts (purposely leaving out behavioral concepts like static or instance methods): (i) it declares the contract of static properties (top-right in Figure 2), (ii) it declares the contract of instance properties (top-left), (iii) it defines values for the static properties (bottom-right), and (iv) instances of that class define values for instance properties (bottom-left). We are currently not explicitly considering methods in our model because (i) in the first step we have no need for adaptive behavior and (ii) we believe that behavior can be well integrated in the approach we are presenting here. In our approach, we are decomposing the three-layered type object pattern into a 2x2 matrix which more closely resembles how class and instance properties are declared and defined in traditional object oriented programming.

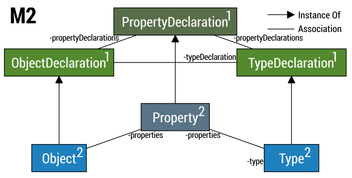

Figure 2: Decomposition of the type object pattern into four elements that correspond to property declaration and instantiation in object oriented programming. Potencies of the types indicate that the top elements are instantiated on M1 and the bottom elements on M0.

3 MODELING APPROACH

In this work we present a structural view on our runtime configurable approach and show how the structural contract of a business model can be modified at runtime without the need for recompilation and redeployment. This structural contract has direct impact on the execution of the business model (mainly by constraining the association of entities and thus reducing the possibilities of configuration).

Specific business models within the REA ontology are traditionally declared in a graphical language based on UML class or object diagram notation[2], which we adopt in the context of this document. They define (i) a taxonomy of entities relevant for that business model (the domain vocabulary) and (ii) the relations between those entities (the business model), e.g. what other entities (mainly agents and resources) certain events interact with or how events relate to commitments.

The remainder of this section will first introduce the concepts we use for the declaration of domain vocabulary and business models (most notable *fragments* and *REA declarations*) and then explain how the declaration is realized.

3.1 Attributes and Associations

`Attributes` are our modeling vehicle to declare variables of built-in data types such as `INT`, `DECIMAL`, and `TEXT`; `Associations` declare complex variables as associations to other `Declarations`. Figure 3 depicts a simplified class diagram of the meta model of the declaration layer M1 in our model. It shows how the REA modeling language is embedded into our model: all REA entities (green: Resource, Event, Agent, ...) inherit from `PropertiedDeclaration`, which essentially means that on the declaration layer, REA entities can be equipped with properties of any kind. Red items (Entity, Declaration, Group, PropertiedDeclaration) represent generic concepts, most notable grouping, cf.(Hrubý et al., 2006). Brown items (Property, Attribute, Association) depict attributes and associations, while blue items (Enumeration, Unit, Fragment) represent additional concepts that will be explained below in more detail.

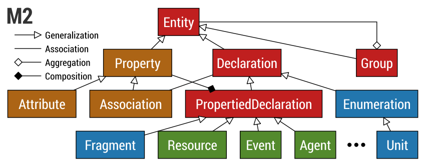

Figure 3: General (simplified) class diagram for M2. Distinct concepts are depicted with different fill colors (explicitly listed and explained in the text). In fact, each of the REA elements (Resource, Event, Agent, ...) is implemented 4-fold, as described in Figure 2, while `Property`, `Enumeration` and `Fragment` are only implemented 2-fold (no typification required).

3.2 Fragments

`Fragments` represent loosely coupled data capsules that, just like core REA entities, inherit directly from `PropertiedDeclaration`, i.e. they can declare named properties (cf. Figures 3 and 4). The purpose of fragments is their reuse in various domain vocabulary declarations or within a single domain. Examples for simple fragments are `HumanName` or `Address`—they have in common that they define rather general concepts that can be used multiple times in the domain vocabulary. In that respect, fragments are very similar to aspects, as described in (Hruby tal., 2006). While the concept of aspects described there is very powerful and even allows the specification of behavior, it is not as smoothly embedded into the modeling layer as our approach, as in our work fragments are first class citizens of the modeling layer and are a standard way of modeling the business vocabulary. In addition, no aspect oriented programming framework is required for usage, and modeling with fragments "feels" just like modeling core REA. Specification of behavior however is one of our elements in the design of a comprehensive software architecture based on the presented data model.

In Figure 4 some generic concepts have been modeled as fragments: `HumanName` and `Address` only specify attributes, while `PersonalInformation` is composed of a `HumanName` and an `Address` *association* amongst two other attributes.

[2]While this makes sense, especially from a software engineering point of view, users of business applications are often non-technical domain experts. For that account graphical concrete syntaxes for domain specific languages have been developed (Mayrhofer, 2012; Al-Jallad, 2012).

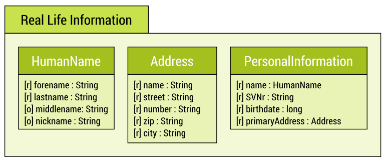

Figure 4: Possible fragment declarations (M1).

3.3 Enumerations, Relations and Units

`Enumerations` inherit from `Declaration`, i.e. they can be associated via associations from fragments or REA entities. Enumerations declare a set of strings which are usually related to each other. Examples are fashion sizes (`XS`, `S`, `M`, `L`, `XL`, etc.) or colors (`red`, `green`, `blue`). On the M1 layer, the enumeration is declared only (i.e. its name is defined), however the values for the enumerations are defined on the M0 layer (cf. Figure 5).

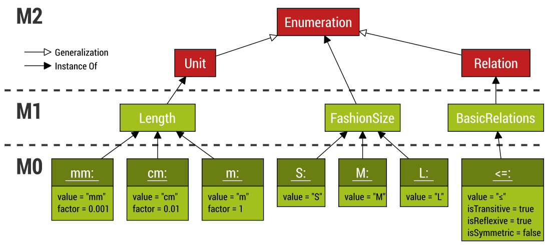

Figure 5: Examples for enumerations, units, and relations.

Units are derived from enumerations and extend them in that they define an additional corresponding factor that relates the values of the enumeration to each other. In the metric system, we can specify length in m, cm, km, etc., in the imperial and US customary systems of measurement length is specified in terms of inch, foot, yard, etc. For the definition of a single unit we therefore need the term needed to describe the unit and the factor that relates the term to the standard unit of the dimension that unit is defined in. Now, at runtime it doesnt matter in which unit a certain value is given, as it can be converted to any other (known) unit easily.

Just like units basically relate different enumeration values by a factor (and thus allow precise conversion from one unit into another), other relations could be defined, too. For instance, the fashion sizes could be related by their size using a "greater than" relation. In fact, the relational information can be modeled completely self-contained, because the relational mappings are defined by a ternary association between enumeration instances: one enumeration instance resembles, the left hand side of the relation, another one the right hand side and a third enumeration instance depicts the operator to be used for the relation. This last enumeration is an instance of the enumeration subclass `Relation` and defines three additional fields: `isTransitive`, `isReflexive`, and `isSymmetric`. Since the operator still is an enumeration instance, its value is of type String—thus the value of the relational enumeration instance could be as simple as "$\leq$", or any literal description.

3.4 REA Declarations

In comparison to fragments that are used to describe domain independent concepts, REA declarations are much more specific, as they implement REA concepts (but still in a domain independent manner when it comes to specific business domains). REA declarations provide the infrastructure to enable (i) the flexible modeling of a specific business domain vocabulary (cf. Section 3.4.1) and (ii) the modeling of the business model (cf. Section 3.4.2).

3.4.1 Domain Vocabulary Declaration

At runtime, REA declarations enable the representation of a business domain in terms of the REA ontology, i.e. instances at the M1 layer are created based on the M2 Meta model infrastructure. These instances on the M1 layer are in turn prescripts for instances on the M0 layer, both of which are managed at runtime. REA declarations are implemented as a set of trees, where each trees root element is one of the core REA concepts, i.e. one tree declares resources, another one declares agents, etc. (cf. Figure 6). In our concept, these declarations define hierarchically structured data capsules only, i.e. each declaration on layer M1 is basically a set of properties (cf. Section 3.1). Behavior is currently not modeled, but support for runtime configurable declaration/definition of methods is planned for the software architecture built upon this data model and thus for future versions of this model.

Each REA declaration is defined by its name, which implies that each name can be given only once. For instance in Figure 6 the given names "Clerk", "Customer", "Sale", etc. cannot be used by any other REA declaration. With this restriction we suppress ambiguities and help designing a cleaner domain vocabulary.

The behavior of a declaration is currently defined by its Meta class only, for instance any kind of agent that is declared on the M1 layer can participate in events, and only the language rules of the REA ontology define the application flow. The provided view on the M1 layer in Figure 6 is a shortcut for what is happening behind the scenes. Based on the power type

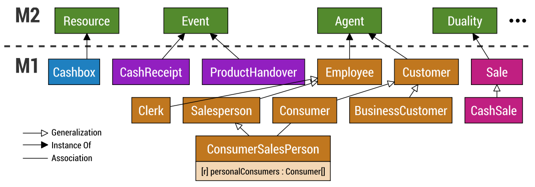

Figure 6: REA declarations: each REA concept spans its own tree of domain vocabulary declarations on layer M1. For the sake of simplicity we exclude further specialization of REA entities, such as the `Event` subclasses `ExchangeEvent` and `ConversionEvent` and their subclasses from this view.

concept, each entity declaration modeled on layer M1 is in the background split in two entities: an instance declaration and a type declaration—the generated data structure is depicted in Figure 7. The instance declaration holds properties declared for each instance of the given entity, whereas the type declaration holds the class properties that are to be defined only once on the M0 layer in a type instance and are referenced from each of the M0 instances through a `type` association. It is declared in the declaration layer whether the properties of fragments and REA declarations are required or optional—thus the runtime engine needs to check object and type instances for the fulfillment of this domain vocabulary peculiarities.

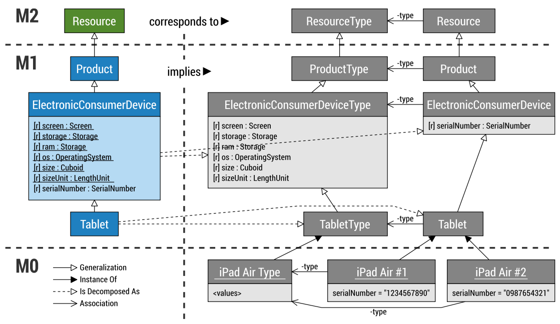

Figure 7: When modeling the domain vocabulary, "behind the scenes" the data structure depicted in grey is generated. The presented properties are all associations, i.e. they either refer to a fragment declaration (this is intended here) or another REA declaration.

3.4.2 Business Model Declaration

Business models are designed at the declaration layer (M1), just like declarations and fragments. For the specification of a value chain of the operational level, the REA declarations need to be arranged accordingly—cf. Figure 8 for a simple sales example, trading products for cash. In this constellation a single sales duality comprises the selling of at least one product (as in a shopping cart) for a single cash payment. At runtime it is rather trivial to check whether an event might occur in a specific duality or not, but of course the inheritance tree must be considered, i.e. instances of any sub-declaration of a REA declaration specified in a constellation are valid entities.

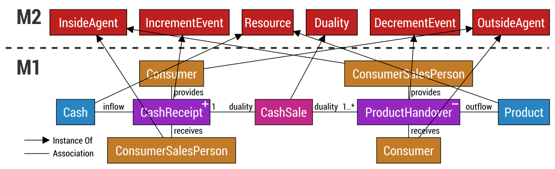

Figure 8: Declaration of a simple "Sales" duality on M1—the cardinality statements on the duality relations specify how often the corresponding event must occur in order to resemble a "valid" duality at runtime (at least one Product must be handed out and only exactly one Cash payment is allowed).

4 CONCLUSIONS

We have presented a data model for the declaration of REA based business models. The concepts of fragments and REA declarations have been sketched and their implementation has been discussed. The presented approach provides a solid basis for the execution of such defined business models in the sense that all required data is available in a single database which can be manipulated and queried at runtime using traditional database interaction methods.

With the separation of declarations and types we have improved the expressiveness of REA models and we have shown that declarations are naturally located on layer M1 while types (of the type object pattern) are really located at layer M0. Our approach provides a consistent view on entities of both layers, and it is thought to be implemented against a single data store, i.e. instances of layers M0 and M1 are "stored together", thus enabling use of referential constraints and referential integrity checking.

We have defined our REA core library with a relational database in the background. Our model provides a class infrastructure for objects and types of the declaration layer M1 and of the runtime layer M0.

We have shown how REA declaration entities can be interconnected in order to resemble business models of varying complexity. With the notion of "fragments", we have introduced a fully integrated new first class citizen in the REA business modeling world. One that does not relate to specific REA concepts but can be applied to any (or none) of them.

One aspect which has not been discussed here is the evolvability of business models, i.e. support for runtime changes in the declaration layer, which is scheduled for further investigation. Also, we are in the process of integrating runtime configurable behavior in addition to the flexible data capsules presented here.

ACKNOWLEDGEMENTS

This work was supported as part of the BRIDGE program of the Austrian Research Promotion Agency (FFG) under grant number 841287—a joint research effort of Vienna University of Technology and eventus Marketingservice GmbH.

REFERENCES

Al-Jallad, M. M. (2012). REA business modeling language: Toward a REA based domain specific visual language. Student thesis, KTH Royal Institute of Technology.

Atkinson, C. and Kühne, T. (2000). Meta-level independent modelling. In *International Workshop on Model Engineering at 14th European Conference on Object-Oriented Programming*, pages 12–16.

Atkinson, C. and Kühne, T. (2001). The essence of multilevel metamodeling. In Goos, G., Hartmanis, J., and Leeuwen, J. v., editors, *UML 2001—The Unified Modeling Language. Modeling Languages, Concepts, and Tools*, Lecture Notes in Computer Science, pages 19–33. Springer.

Atkinson, C. and Kühne, T. (2008). Reducing accidental complexity in domain models. *Software & Systems Modeling*, 7(3):345–359.

Frank, U. (2011a). The MEMO meta modelling language (MML) and language architecture. ICB-Research Report 43, Institute for Computer Science and Business Information Systems, University Duisburg-Essen.

Frank, U. (2011b). Some guidelines for the conception of domain-specific modelling languages. In Nüttgens, M., Thomas, O., and Weber, B., editors, *4th International Workshop on Enterprise Modelling and Information Systems Architectures (EMISA 2011)*, volume P-190 of *Lecture Notes in Informatics*, pages 93–106, Bonn. Gesellschaft für Informatik, Köllen Druck+Verlag GmbH.

Gailly, F. and Poels, G. (2007). Towards ontology-driven information systems: Redesign and formalization of the REA ontology. In Abramowicz, W., editor, *Business Information Systems*, volume 4439 of *Lecture Notes in Computer Science*, pages 245–259. Springer Berlin Heidelberg.

Gailly, F. and Poels, G. (2010). Conceptual modeling using domain ontologies. improving the domain-specific quality of conceptual schemas. In *10th Workshop on Domain-Specific Modeling*, pages 18:1–18:6. ACM.

Geerts, G. L. and McCarthy, W. E. (1997). Modeling business enterprises as value-added process hierarchies with resource-event-agent object templates. In Sutherland, J., Casanave, C., Miller, J., Patel, P., and Hollowell, G., editors, *Business Object Design and Implementation*, pages 94–113. Springer London.

Geerts, G. L. and McCarthy, W. E. (2000). The ontological foundation of REA enterprise information systems. In *Annual Meeting of the American Accounting Association, Philadelphia, PA*, volume 362, pages 127–150.

Geerts, G. L. and McCarthy, W. E. (2006). Policy-level specifications in REA enterprise information systems. *Journal of Information Systems*, 20(2):37–63.

Gürth, T. (2014). Business model driven ERP customization. Master's thesis, Faculty of Informatics, Vienna University of Technology.

Hrubỳ, P., Kiehn, J., and Scheller, C. V. (2006). *Model-Driven Design using Business Patterns*. Springer.

Johnson, R. and Woolf, B. (1997). Type object. In Martin, R. C., Riehle, D., and Buschmann, F., editors, *Pattern Languages of Program Design 3*, chapter Type Object, pages 47–65. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Kühne, T. and Steimann, F. (2004). Tiefe charakterisierung. In Rumpe, B. and Hesse, W., editors, *Modellierung 2004*, volume P-45 of *Lecture Notes in Informatics*, pages 109–120, Bonn. Gesellschaft für Informatik, Köllen Druck+Verlag GmbH.

Mayrhofer, D. (2012). *REA-DSL: Business Model Driven Data Engineering*. PhD dissertation, Vienna University of Technology.

Mayrhofer, D., Mazak, A., Wally, B., Huemer, C., and Regatschnig, P. (2014). REAlist: Towards a business model adapting multi-tenant ERP system in the cloud. In *8th International Workshop on Value Modeling and Business Ontology (VMBO 2014)*.

McCarthy, W. E. (1982). The REA accounting model: A generalized framework for accounting systems in a shared data environment. *The Accounting Review*, 57(3):554–578.

Nakamura, H. and Johnson, R. E. (1998). Adaptive framework for the REA accounting model. In *OOPSLA'98 Workshop on Business Object Design and Implementation IV*.

Object Management Group, Inc. (2013). *OMG Meta Object Facility (MOF) Core Specification*. Object Management Group, Inc.

Wally, B., Mazak, A., Mayrhofer, D., and Huemer, C. (2014). A generic REA software architecture based on fragments and declarations. In *8th International Workshop on Value Modeling and Business Ontology (VMBO 2014)*.

Yoder, J. W. and Johnson, R. (2002). The adaptive object-model architectural style. In *Software Architecture*, pages 3–27. Springer.