A Generic REA Software Architecture based on Fragments and Declarations

Bernhard Wally, Alexandra Mazak, Dieter Mayrhofer, and Christian Huemer

Vienna University of Technology, Institute of Software Technology and Interactive Systems, Business Informatics Group, Favoritenstrasse 9–11, 1040 Vienna, Austria,
{wally, mazak, mayrhofer, huemer}@big.tuwien.ac.at,
http://www.big.tuwien.ac.at/

Abstract. The implementation of a domain agnostic software system (such as an enterprise resource planning system or a trading information system) based on the REA model requires a generic, yet highly adaptable software architecture. Building on top of well established academic findings and proposals regarding the REA model we are proposing a software model building on two main concepts: *fragments* and *declarations*. Together they establish a single, flexible, extendible class hierarchy implemented purely on a database layer. Fragments and declarations are implemented as associative arrays that are enriched with metadata, providing clear access to attributes and their meta information. The system's flexibility is drastically increased by enabling the multi-typification of REA entities.

Keywords: Enterprise Information Systems, Business Ontologies, Software Engineering

1 Introduction

The REA model has been investigated on an abstract level [9,4,3], in terms of software engineering approaches [10,6,7] and some attempts have been made to implement real-life REA based business software in the past [6]. Some authors have researched on rules and policies for the REA model [1,5,2] in order to provide advanced runtime configurations, constraint definition and checking, etc. In our software architecture for the REA model we are seizing some of those approaches and adapt them slightly to define a model that suits our needs and feels very generic from a software engineering point of view. Our model is backed by a domain independent data structure and provides a mechanism for the configuration and execution of business models at runtime.

We also make heavy use of the "type object" pattern [8], which has been used for describing REA models in the past. With respect to the typification method in REA, we are using this pattern to also enable the possibility for runtime-specification of more than one type on a single instance. We thus present a technical solution to the multi-typification issue which, in our opinion, has not received enough attention in the past.

2 Runtime Modeling

It is an important aspect of long running systems to be able to adapt over time—in order to accommodate changes in the environment, evolving user preferences, etc. In case of an accounting model such as REA [9] this means to be able to host varying and changing businesses with different requirements, entities and business models. The traditional way in REA to model a certain business is to declare it on a "type" and on an "instance" layer and use associations among entities of both layers to define relationships and policies [5]. [6] extends this concept by the notion of "aspects", i.e. horizontal data and function capsules that can be woven into virtually any domain concept by using methods of aspect oriented programming.

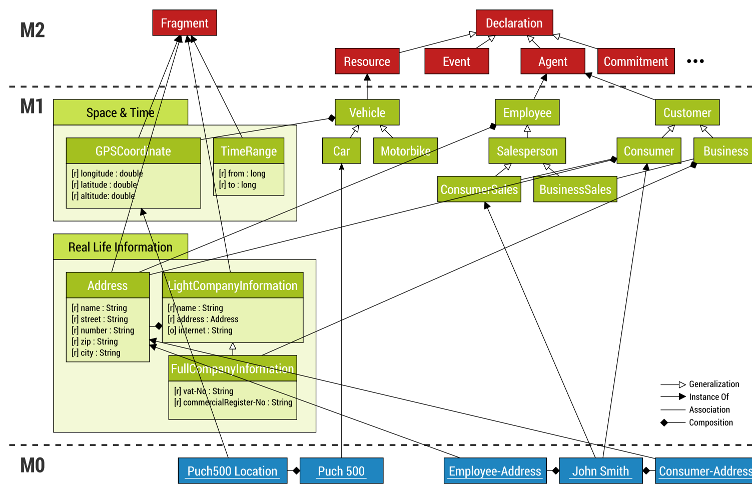

Fig. 1. Overview of our class hierarchy (mix of class diagram and object diagram)—the left part of the diagram displays "fragments", the right part depicts the inheritance tree of the "declarations". Classes depicted in red are core classes, representing domain independent entities of the REA model (not all REA concepts shown here) or other entities situated at the REA model level; these classes are implemented in software. All classes depicted in green are examples for those that are declared at runtime, backed by a generic database, basically as associative arrays including metadata, i.e. they define their named attributes and attribute types, and whether these attributes are required or optional. Blue items depict specific objects (instances) of green class declarations, and they can be realizations of multiple class declarations in a single instance (multi-typification).

In our work we are reshaping things a little bit by making fragments first class citizens of the REA modeling layer and by providing a generic solution for the issue of multi-typification. We are introducing a database backed Meta-

Object Facility[1] (MOF) compatible modeling scheme. In Fig. 1 we provide an overview on our global software architecture by displaying a tiny domain specific example.

We are using the REA model plus "fragments" as language definition on layer M2, we are defining the domain specific business models on layer M1 and we are instantiating the busines model entities on layer M0.

- The M2 layer is defined by the REA model in terms of an UML compatible class diagram, plus we add a little modeling sugar by injecting the concept of fragments to it.
- The M1 layer is used by business domain experts to declare their business model by the notion of domain specific resources, events, agents, etc. Entities defined on this layer correspond to what is called an "interface" or "pure virtual class" in software engineering terms: they declare their public methods—in our case the indices of the underlying associative array.
- The M0 layer represents instantiations of M1 entities, i.e. the real resources, events, agents, etc. It is crucial to note that each entity on the M0 layer can be associated with multiple M1 entities, i.e. an agent can be both an employee and a customer. Of course, the runtime instance must fulfill the class declaration contract defined by all the M1 entities it is associated with. In our current concept, a single M0 entity can be associated with at most one kind of REA model entity, i.e. it is either a resource, or an agent, etc. but within such a category it can be associated with multiple declarations thereof.

2.1 Fragments

Fragments are reusable data capsules (i.e. in our current concept they do not expose or possess any behavior) with the following main properties:

- Fragments are sets of attributes with a common context, such as e.g.:
  - `Identifiable` (providing a string value to store unique identifiers),
  - `Schedulable` (providing values for the duration and for the preparation and wrap-up time), or
  - `GPSCoordinate` (providing values for longitude, latitude and altitude).
- They are flexible units that can be referred to by declarations, in order to declare their capabilities in a generic and reusable way, and
- they are declared scope-free, i.e. they can be applied to both types and instances.

Fragments define their own class hierarchy, with a common base class `Fragment`, which in turn is a direct descendant of `Entity`. A single fragment can inherit explicitly from at most one other fragment—if it does not, it implicitly inherits from

[1] http://www.omg.org/mof/

`Fragment`. A fragment can also define identifiers referring to other fragments by declaring a "composition" link, e.g. the `CompanyInformation` fragment might define a property `location` which refers to a `GPSCoordinate` fragment.

2.2 Declarations

In order to distinguish REA concepts from our chosen software engineering approach we have decided to name REA entities "declarations", as they might be more restrictive in some cases and more flexible in other cases, than REA itself—it is simply easier to refer to "REA concepts" on the one hand and to "declarations" on the other hand.

Declarations offer a system for tree-based modeling of domain specific characteristics with the help of a number of parallel trees: there exists a tree for declaring agents, another tree for declaring resources, another one for declaring events, etc. Reduced to its semantics, declarations correspond to the core REA model, i.e. specific business domains are modeled by specifying resources, agents, events, etc. For our approach it is however important to formalize REA concepts in software engineering terms; as such the term "declaration" stands for a class declaration just like in object oriented languages, i.e. declarations define a class hierarchy and offer variables for instances and for classes (the latter correspond to what is usually called "types" in the REA model). Each node ("declaration") of the tree can be associated with zero or more fragments by realizing a "composition" link, thus declaring the structured attribute contract the types and instances of the runtime layer must adhere to.

Also, declarations can have associations with other declarations, e.g. an aircraft maintenance technician can have the permission to maintain specific aircraft types. In addition, such an association can have an association class attached to it, and that class must be an instance of a fragment. For each instantiation of such an association, a separate instance of the corresponding association fragment is created and linked as an instance of the association class.

3 Conclusion

We have presented a generic software architecture for an REA based accounting system that uses a variant of the type object pattern as the core modeling technique in order to allow separation of domain specific knowledge from the software model. Our approach allows modelling on two interlinked parallel paths: fragments and declarations. The latter make use of the former by composition links as instance or class variables. A clear separation between the modeling layer and the runtime layer is established, and on the runtime layer each instance can be typified with multiple declarations, which enables a high grade of flexibility for the runtime system. The class contract of fragments and declarations further allows the structured definition of business rules on both the modeling layer and on the runtime layer; through attribute and metadata inspection graphical user interfaces can provide thorough assistance in the formulation of business rules.

We are currently evaluating our software architecture for usability (from a software engineering point of view) and applicability with regards to the integration of business rules.

4 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

1. Andersen, J., Elsborg, E., Henglein, F., Simonsen, J.G., Stefansen, C.: Compositional specification of commercial contracts. International Journal on Software Tools for Technology Transfer 8(6), 485–516 (October 2006)
2. Gailly, F., Geerts, G.: Formal definition of business rules using REA business modeling language. In: 7th International Workshop on Value Modeling and Business Ontology, Proceedings. p. 7 (2013)
3. Gailly, F., Poels, G.: Towards ontology-driven information systems: Redesign and formalization of the REA ontology. In: Abramowicz, W. (ed.) Business Information Systems, Lecture Notes in Computer Science, vol. 4439, pp. 245–259. Springer Berlin Heidelberg (2007), http://dx.doi.org/10.1007/978-3-540-72035-5_19
4. Geerts, G.L., McCarthy, W.E.: The ontological foundation of REA enterprise information systems. In: Annual Meeting of the American Accounting Association, Philadelphia, PA. vol. 362, pp. 127–150. Citeseer (2000)
5. Geerts, G.L., McCarthy, W.E.: Policy-level specifications in REA enterprise information systems. Journal of Information Systems 20(2), 37–63 (Fall 2006)
6. Hrubỳ, P., Kiehn, J., Scheller, C.V.: Model-Driven Design using Business Patterns. Springer (2006)
7. Huňka, F.: Power-types in business process modeling. Journal of Applied Economic Sciences (JAES) 8(1 (23)), 52–62 (Spring 2013)
8. Johnson, R., Woolf, B.: Type object. In: Martin, R.C., Riehle, D., Buschmann, F. (eds.) Pattern Languages of Program Design 3, chap. Type Object, pp. 47–65. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1997), http://dl.acm.org/citation.cfm?id=273448.273453
9. McCarthy, W.E.: The REA accounting model: A generalized framework for accounting systems in a shared data environment. The Accounting Review 57(3), 554–578 (July 1982)
10. Nakamura, H., Johnson, R.E.: Adaptive framework for the REA accounting model. In: Proceedings of the OOPSLA'98 Workshop on Business Object Design and Implementation IV (1998)