Object oriented REA using DBQUITY

Jesper Kiehn[1] and Lars Hammer[2]

[1] *EG A/S, Dbquity ApS, Copenhagen, Denmark*
[2] *Hammer Software, Dbquity ApS, Frederiksberg, Denmark*

1. Introduction

In this presentation, we introduce the software platform, Dbquity [9], and some learning points about REA models.

Dbquity lets you declaratively model data structure and express derived information at the domain expert's level of abstraction without over-specifying or repeating knowledge using a dedicated declarative1 language, which centers on the notions of entity and association and single inheritance with powerful member-specialization capabilities, and lets you work on multiple levels of abstraction factoring out domain-specific, reusable libraries.

The language abstracts away all implementation details, and the model declared is executed directly by the Dbquity runtime without any need for further artefacts.

Design goals for Dbquity include, that based on a single inheritance combined with expressive, intuitive constraints, the language must support both basic and more advanced REA patterns such as policy and valuation whilst aiming to be if not writable, then at least readable for non-programmers using a no/low-code approach. Further, the system must be able to generate the required set of reports from the model itself in as simple a way as possible.

We aim to open source the specification of the language itself and keep the core tooling and runtime implementations closed source. The first2 incarnation of the runtime targets a simple, ubiquitous two-tier topology using cloud-based storage and mobile phones aiming to make it as effortless as possible for modelers to distribute Dbquity models and for consumers to use the models.

Our work includes several examples to make it clear how to implement REA patterns using Dbquity, and we hope to get feedback on the presented models, the language, the tooling, and the current scope.

1.1 The Dbquity language introduced by an example

The example of (a sketch of) a REA library and a DAAS design referencing that library in figure 1 on the following two pages shows how the Dbquity language uses semantic indention such that text indented under a model element is either properties of that model element or nested elements or members of that same model element, meaning that the `Agent` entity will have a field of type text called `Name` and the identity of the `Agent` will be the `Name`.

In general, a type followed by a name declares an element or a member of that type, and properties are declared by their name followed by a colon and the property value, which is often an expression over the model.

Dbquity stores data in a hierarchy of entities rooted by areas. An area is itself a (special kind of) entity. Only concrete (not abstract) entities declared inside an area will be instantiable at runtime.

Proceedings of the 16th International Workshop on Value Modelling and Business Ontologies (VMBO 2022), held in conjunction with the 34th International Conference on Advanced Information Systems Engineering (CAiSE 2022), June 06–10, 2022, Leuven, Belgium
EMAIL: jkiehn@hotmail.com (J. Kiehn); lars.hammer@dbquity.com (L. Hammer)

[1] While continuing to strive for maximum declarativeness, we have not – yet? – been able to avoid all and any imperativeness, and the language seems to still need some imperative constructs, e.g., for adding an entity and setting field values in behavioural expressions.
[2] Other runtime implementations capable of executing the exact same Dbquity models – for example webserver-based ones, as well as some that focus on scalability in terms of concurrency and data amounts – are anticipated.

Note how the name of a previously declared entity can be used as a keyword for declaring a specialized entity. For example, the line `Agent Producer`, declares the `Producer` entity in the `Sales` area specializing the `Agent` entity of the `REA` library.

```
library REA
  entity Resource
    text Name
    identity: Name
    decimal Price

  entity Event
    date When
    decimal Amount

  entity Agent
    text Name
    identity: Name

design DaaS
  references: REA

  area Sales
    Agent Producer

    Agent Lessor
      decimal ComputerPurchaseTotal
        expression: Lessor@Sale.sum(Amount)
      integer ComputersBorrowed
        expression: Lessor@ComputerBorrowSale.count(Active)

    Agent User
      link Employedby
        entity: Lessor

    Resource Computer
      link User
        expression: Computer@ComputerBorrowSale.last(Active).User
      link Soldto
        entity: User
        expression: Computer@ComputerBorrowSale.last(Sold).User

    Resource TransportService

    Event Sale
      link Producer
      link Lessor
      link Computer

    Event SalePayment
      link Producer
      link Lessor
      link Sale

    Event Transport
      link Producer
      link Lessor
      link TransportService
      link Computer
```

Listing 1: Example of a Dbquity design referencing a library (continues on next page)

```
Event ComputerBorrowSale
  link Lessor
  link User
  link Computer
  date LatestReturn
    default: When.addyears(3)
  boolean Active
    expression: not (Returned or Sold)
  boolean Returned
    expression: ComputerBorrowSale@Return.any()
  boolean Sold
    expression: ComputerBorrowSale@UserPayment.any()
  step IssueNotice
    guard: today() >= LatestReturn.adddays(-10)
    behaviour:
      add(Notice,
        Due: max(LatestReturn,today()),
        Amount: Amount,
        Note: "Please return PC by "|max(LatestReturn,today())|
          " or pay "|Amount|"USD.")

  entity Notice
    decimal Amount
    date Due
    text Note
      multiline

Event Return
  link ComputerBorrowSale

Event UserPayment
  link ComputerBorrowSale
```

Listing 1: Example of a Dbquity design referencing a library (continued from previous page)

2. References

[1] William E. McCarthy, G. L. Geerts "An ontological analysis of the economic primitives of the extended-REA enterprise information architecture"
[2] William E. McCarthy, G. L. Geerts: "Policy-Level Specifications in REA Enterprise Information Systems"
[3] Valueflows: https://www.valueflo.ws/
[4] Cheryl Dunn , J. Owen Cherrington, Anita Hollander: Enterprise Information Systems: A Pattern-Based Approach
[5] Cheryl Dunn: REA Accounting Systems
[6] Frederik Gailly; Wim Laurier; Geert Poels: Positioning and Formalizing the REA Enterprise Ontology
[7] James Perry, Richard Newmark: Building Accounting Systems Using Microsoft Access 2013
[8] ISO/IEC 15944-4
[9] http://www.dbquity.com/
[10] https://grpc.io/
[11] Hruby, Pavel. (2006). Model-Driven Design Using Business Patterns. 10.1007/3-540-30327-2.