

A Property Driven Approach Towards Creating an Adaptive REA Framework

Mette Jaquet and Kasper Østerbye
IT-University of Copenhagen
Software Development Group
{ mjaquet, kasper }@itu.dk

Presented at the REA-25 Conference, June 13-15, 2007, Delaware

Abstract

Since it was introduced 25 years ago, the Resource Event Agent model has gained attention as a promising proposal for a universal model of economic phenomena in enterprises. While being flexible and generic, the informal nature of the REA ontology leaves important issues open for interpretation, resulting in rather heterogeneous business models and implementations. We propose a generic meta-model of event-driven systems in general and REA in particular, that preserves flexibility, allowing heterogeneity through loosely coupled properties. This paper describes an implementation inspired by the Active Object Model combined with a Property-driven approach. Thus allowing the addition of not only entity types, groupings and business rules, but also new relations and attributes of entities to data of running systems. Application of such a common meta-model would give existing systems extensive flexibility while enforcing consistency, and ease model evolution and integration with other systems significantly.

Keywords: REA Model, Business Ontologies, Enterprise Framework, Event-driven Systems, Interoperability, Active Object-Model, Adaptive Systems, Meta-modeling

I. Introduction

The REA-model describes events affecting the value and flow of resources within and between enterprises. When deciding what to model and how to model it there are two key issues. What are the *entities* we want to observe, and what are the *changes* affecting these that we want to record? As the intent of the REA-model is to describe economic phenomena of enterprises, the main entities we want to monitor besides agents and agreements are resources and their value. Determining the value of a resource may be done in many different ways, depending on i.e. the cost of production or purchase, or on its "want-satisfying power" [Black & Black, 1929] determining how much a customer is willing to pay for it. [Geerts & McCarthy, 1999] states that "*Value is defined as a deliverable portfolio of product or service attributes attractive to the firm's ultimate customers.*". So besides from exchanging resources with customers and vendors, changes to the attributes or properties of resources can affect the overall values within an enterprise. Thus, we need to know *which entities* the enterprise has control or possession of and *what state* they are in.

That leads to the second key issue of creating REA models. What changes are significant to model? As stated in [Jaquet, 2006] deciding which value-affecting properties of resources are significant to model, may determine which events to model. The events of interest are those that add or modify those properties. In this sense property changes may be either physical transformations or modified relations of i.e. ownership or control.

The REA model aims at providing a basic generic model that can describe economic phenomena of several different systems, both within and between enterprises of many different types. Furthermore a specific business model of an enterprise system evolves during its lifetime, and new types of resources or new properties of existing resources may be introduced. Thus an important success criterion for an enterprise system, and the architecture of its underlying framework and domain model, is the flexibility to allow such changes to be introduced as seamlessly as possible. One way of achieving such flexibility is to base the architecture on a generic meta-model as i.e. the Active Object Model [Yoder et al., 2001]. In this paper we will present the design of an adaptable framework for event-driven systems like those based on REA, using an AOM-inspired architecture. We will not try to describe all parts of such a system in detail here, but a generic model with a notation for describing the data format will be presented for the basic REA entities, and extended

to provide complete flexibility when using the property-driven approach of maintaining loosely coupled attributes.

The basics of the REA model will not be mentioned in this paper. Any readers without prior knowledge of the REA ontology may find an overview in [Geerts & McCarthy, 2000] or a more elaborate description aimed in particular at model-driven development of enterprise systems in [Hruby et al., 2006]. Several variants of the ontology definition exist. In this paper we will adopt the naming of concepts as described in [Hruby et al., 2006] and shown in Figure 1.

*** Insert Figure 1 here ***

A traditional approach to developing a REA model involves determining what processes take place in an enterprise and breaking these down into recipes of tasks and events [Geerts & McCarthy, 2000]. The REA ontology describes the relations between entities rather detailed, but leaves the attributes of REA entities as an implementation issue. We argue, however, that being able to describe the properties of REA entities in a flexible yet uniform manner is a key to achieving adaptable and interoperable enterprise systems.

As an ongoing example throughout this paper we will use a small enterprise called Andy's Auto's. Andy's Auto's started out selling used cars, but evolved to also include a garage performing repairs and repainting, and ended up also having cars for rent. Such modifications of the business model, requires introducing new properties (both relations and attributes) to existing entities. Besides from changing the data-structure, new types of queries may also be required. Conclusion of materialization is often based on the *ownership* relation - calculating balances of what is currently owned by a company. But *rights of use* and other agent-resource relations might be equally relevant to derive values from. Properties may also serve as the base of queries or conclusions. In order for Andy's Auto's to determine if a car is ready for rental, it may not be sufficient that the *rights of use* relation and physical control of the car is reestablished after a car has been returned by the previous renter. The car may also require an *inspection* event, updating the state of an *inspected* property, making the car ready for rental again.

Creating an operational system driven by state changes affecting properties of entities, is event-driven by nature - although not necessarily only by the economic events the REA model describes.

With this paper we describe some of the prerequisites and issues to consider when designing a framework for such a system.

II. An Event-Driven Framework

The REA model is event-driven by nature, recording basic facts about economic changes and leaving values, as i.e. the current balance of an account, to be derived. Using derived values calculated on request by summation of recorded increment and decrement events, gives the possibility of adding retrieval of additional values from existing data to a running system, as opposed to maintained values that are continuously updated to reflect the current value, and must be implemented when designing the system. The difference may be seen as applying the filter used for obtaining the relevant data at the time of information retrieval rather than the time of information recording. Ease of sharing and reusing data, as well as integrating systems are also among the several benefits of basing enterprise systems on an event-driven architecture. According to [Hoppe, 2006] there are Enterprise Application Integration vendors proclaiming "*that event-driven architectures are the next step in the evolution beyond Service-oriented Architectures (SOAs)*". We aim at establishing a notation for describing data of a REA-system and recording the events that change these data. But before we approach the definition of an event we will establish a number of more fundamental concepts.

System boundaries

Any system is a part of the world, and as such has a boundary. The parts inside the boundary are of interest; those outside the boundary are not. In a designed system, designing the boundary is an important part of the design. The boundary is not a necessarily a physical delimitation, but is often one of relevance. In a system for Andy's Auto's the boundary would depend on the way business is run. If Andy calls in a specialist to perform rust treatments of his cars, he may buy and consume a *rust treatment* service and the materials, equipment and labor used to perform it are not relevant for him to record. If he performs it himself these entities will however need to be included the system. Besides from determining what entities to include in the system, the boundary may also affect what attributes of the entities it is relevant to observe. I.e. if cars are sold their color will be recorded,

but if they are repainted the exact formula may be required as well. We shall only consider aspects within the boundary.

An enterprise system has both a real-world and a computerized part to it. In [Jackson, 1983] Michael Jackson terms the "*real world with which the system is concerned*" Level-0 and the "*direct model of the real world*" Level-1. Following this terminology our first approximation to a definition of events will then be:

An event is a change of state in Level-0.

This immediately raises the issue of state and change. To talk about state and state change, we will adapt the definitions by Kristen Nygaard [Madsen et al., 1993].

A Level-0 system consists of objects with measurable properties. The result of a measurement is a value. Objects are connected through relationships. Relationships have no measurable properties. Transformations change either the measurable properties of objects or relationships between objects.

Using the terminology of Sowa [Sowa, 2000] objects can be seen as continuants while transformations are occurrents. The real-world part of the system consists of continuous processes, involving sequences of Level-0 transformations that are either discrete or continuous. However, the computerized Level-1 operates with only discrete transformations. Relating Level-0 transformations to Level-1 thus becomes part of the modeling task.

To address the issue of continuous transformations, we introduce an observer. An observer examines the objects and their relationships in Level-0 and reports Level-0 transformations as Level-1 *events*. A *discrete observer* looks for changes to specific level-0 objects or their relationships. When such a change happens, an event is reported to Level-1. A *scheduled observer* reports selected measurements at regular intervals. Thus, observers produce an event-history which forms the basics of an information system.

For now, we will reserve the term *event* for something being reported by an observer in the Level-1 event-history. State and relationship changes at Level-0 is done through object *transformations*.

Often observers are human agents reporting real-world events through a user interface. But also automatic sensors or other computerized systems may report events. I.e. scanning of RFID tags may report changes in the location of resources.

Types of Events

While the paper-less society is still to come, a number of business artifacts are today fully computerized, and the relationship between Level-0 and Level-1 is somewhat reversed. Stocks, bonds, and home-banking are examples. A monthly transcript from the bank represents a physical representation of the real digital record. Jacob W. Jespersen [Jespersen, 2006] defines a differentiation between three kinds of relationships connecting Level-0 and Level-1.

- **Simulation** is the situation described above, where Level-1 is a model of Level-0.
- **Augmentation** describes the situation where computer systems augment the real-world, that is, the objects represented at Level-1 has no counter-part in Level-0, but have become part of the discourse in Level-0. Email, home pages, etc. are well-known examples.
- **Transposition** covers objects whose Level-0 substance is replaced by its Level-1 substance. Stocks, bonds are examples.

In the above, events were defined for simulation. However, augmentation and transposition is digital in nature, and all state and relation changes are discrete, and may be *logged* to an event-history. This gives us a uniform way of handling all three types of relationships between Level-0 and Level-1. The term *recording* will be used to cover both Level-0 *observation* and Level-1 *logging*.

The concept of "event" in REA is that of economic events. The main purpose is to register such events, and relate them in dualities to track the exchanges done by the enterprise. However, a number of papers have over time proposed the addition of other kinds of events, e.g. *business events* and *information events* as described by i.e. [Denna et al., 1993] and [David et al., 2000].

Our understanding of *event* extends the REA *event* concept beyond economic events as well. If we, as a start, include the basic REA entities in our system, then resources, agents and transformations of their properties and relations are Level-0 aspects. The traditional REA events are events reported regarding either changes of ownership relations between agents and resources, or physical transformations of resources. However, our understanding of events (as significant state changes in the Level-0 part of the system) dictates that also changes to agents, and relationships between agents need to be recorded as events in an operational REA framework. Examples of such events

might be employees being hired, promoted, resigned, re-located or having planned absence due to i.e. vacation, education or maternity-leave.

While updates to the state and relations of resources take place through transfers and transformations, other relations of the REA model also need to be updated in an operational model. As an example, the *fulfillment* and *duality* relations connecting commitments with events and events with events, respectively, need to be recorded. These settlements may either be performed manually by human users or aligned automatically, but either way the event must be recorded. Adding new types, instances or groupings of agents and resources are also events that must be recorded and handled on the fly by an adaptive system.

Information events of relevance can often be described as state changes of existing objects. In the case of Andy's Auto's, updating a "Drivers License shown" property of customers renting cars may be required with regular intervals. Traditional examples may be sending an invoice, that is not an economic event, but may change the state of an order to "invoiced" affecting i.e. the date payment is due, or sending a quote that may be followed by receipt of a signature or other form of acceptance from the customer, thereby introducing a contract.

Observing Events

As mentioned above, determining the boundary of a system is a question of deciding which real-world entities are of interest, which relationships are of interest, and what properties of these objects are of interest.

The definition of such a boundary, describing what is part of the domain model and what is not, also becomes the "set of glasses" used by the observers that provide the interface between Level-0 and Level-1. The description may be formal and used by an electronic device determining what should be logged, or it may be the less rigid conceptual model of a user entering data through a user interface. Either way the model used by observers determines the amount and format of data from real-world processes available at Level-1.

*** Insert Figure 2 here ***

Figure 2 shows how the flow of information through the system typically forms the following cycle:

1. State changes observed by users or reported by sensors or other systems are recorded as discrete events forming an event-history. Note that some events may be recorded at a different time than they took place, and both the time of occurrence and the time of recording may be of interest. All data present in the system comes from the event history, so only the information included by the observer's model of the business is available.
2. Based on data from the event-history and generic or specific business logic, conclusions of materialization can be performed by a conclusions engine, deriving current values of i.e. account balances, state of resources, staff available etc.
3. The resulting reports and answers to queries, can be presented to users through a graphical user interface (GUI) or forwarded to other systems.

Further events may be triggered by responses from users or other systems, constraints on values, temporal rules and so forth. These events are recorded in the event-history, repeating the cycle all over again.

The domain model or ontology of the system is represented several places, in heterogeneous formats. A user's mental model, a sensors formal model, a conclusions engines computational model and a GUIs representational model may differ in format but must share the same semantic model. Thus an important criterion of a successful implementation is creating and maintaining consistent versions of this model, allowing all parts of a system to adapt to a changing business environment.

III. Designing a Framework

The issue addressed here is not that of heterogeneous business models but heterogeneous representations of the same model. We therefore aim at providing a common meta-model allowing heterogeneous representations and flexible modifications. In [Nakamura & Johnson, 1998] Nakamura and Johnson show that the Active Object-Model seems particularly well suited to provide adaptability and represent the necessary evolvability of resource and agent types. However, at the heart of REA

is, we believe, events. We will present an approach extending AOM, to include also evolvability of events changing the state of objects and the attributes they change.

The Active Object-Model

Some of the benefits of using an Active Object-Model (AOM) approach [Yoder et al., 2001] when designing a system, are the possibilities of changing types of existing object instances, as well as adding new types on the fly. In an AOM system, classes, attributes, and relationships are represented as metadata, basing the object-model on instances rather than classes. The object-model is thus interpreted at runtime rather than being fixed before deployment, and may be altered by users to reflect changes in the domain. The object-model is called *Active* since updates to the model apply instantly, changing the behavior of the system. The disconnection of the object-model and behavior of a system requires a high level of abstraction when implementing such a generic system. The framework should not reflect the business model, but be a machine capable of executing a business model. The approach is often used when i.e. managing products, requiring new types of resources and perhaps also production methods to be introduced regularly.

The central concept of AOM is the Type-Object pattern [Johnson & Wolf, 1998] relating instances of Entities to instances of EntityTypes. The separation between Knowledge Level and Operational Level, as proposed by Martin Fowler in [Fowler, 1997] and applied in the REA ontology [Geerts & McCarthy, 2000] is an example of the Type-Object pattern. While the REA ontology does not address the attributes of entities, AOM defines Entities as having attributes that may be described by the Property pattern [Foote & Yoder, 1998]. The Type-Object pattern can then be used again to separate Properties from PropertyTypes giving the TypeSquare figure shown in Figure 3.

*** Insert Figure 3 here ***

When all types of objects are instantiated from a single class, their attributes can not be implemented using instance variables. Instead Entities may have an instance variable holding a collection of Properties describing its attributes. If Properties have a PropertyType, then an EntityType can specify which PropertyTypes its instances may have. The Properties described may be either Attributes having a value or Associations having a reference to another object. As an added benefit,

the AOM thereby allows a perfect fit with the property-driven approach to modeling adaptive REA-systems as described in [Jaquet, 2006]. The Property pattern eliminates the need for subclassing when it comes to attributes, but different kinds of objects usually also have different kinds of behavior. This cannot be reflected using Properties and [Yoder et al., 2001] suggests using the Strategy pattern representing algorithms for this.

AOM gives the possibility of adding new entity types on the fly, as well as altering, adding or removing both types and properties of existing entities. We thus believe that the Active Object-model seems a suitable way to represent the underlying data structure of our framework ensuring adaptability. As a first step towards an adaptive framework, we proceed to describe how an instance of such a data structure can be created and populated by recording events forming an event-history. How to render and maintain the data structure and business rules through a graphical user interface will not be discussed in this paper, [Yoder et al., 2001] describes an example of an approach. Likewise, how querying and reporting conclusions is supported by AOM, lies beyond the scope of this paper. The topic is discussed to some extent in [Nakamura & Johnson, 1998].

Recording an Event-History

So what needs to be recorded in the event-history of an enterprise system? As described earlier we want to record objects that are *added* and objects that are *modified*. Since the event-driven approach of REA relies on maintaining a complete history of events, no objects are deleted. For practical reasons this may be desirable in real implementations, but we will disregard the possibility for now.

Events introduce or modify certain kinds of objects. In enterprise systems these may be the familiar REA entities, namely instances of *agents*, *resources*, *commitments* and *contracts* or *type images* of these. Furthermore the entity *relations* of REA and *locations* as described in [Denna et al., 1993] and [Hessellund, 2006] may be added or changed.

All events taking place affect at least one specific instance of an object and thus need to record a unique *object id* referencing i.e. an instance of a resource, agent or contract.

Since we have defined events as being discrete, all events have a single *time of occurrence* as well as a *time of recording*. Every event also has a unique id to be referenced by, i.e. when settling duality relations.

The different types of REA events, such as transfers and transformations, may be characterized by the properties they change. The typical transfer changes the value of an *ownership* relation from one agent to another. The use of equipment may change the state of its availability and raw materials that are not individually discernible, like fuel, changes the amount of that resource type at a specific location. These are the types of events traditionally modeled as REA events. But as mentioned earlier also other relations of the REA model than those involving stock-flow of resources may be updated and need to be logged in the event-history. For instance the fulfillment of commitments (creating fulfillment relations between events and commitments) and pairing of dual events (creating duality relations between events) are significant events, that may be initiated either manually by a user or through automatic alignment procedures, but either way needs to be recorded.

An operational REA model will also need to allow new instances to be created of entities, like i.e. agents and contracts. As mentioned previously the event-history should also include relevant information events, which often can be seen as state changes to existing objects. An example may be the need for recording that an invoice has been sent. This may be modeled as an update to the "invoice sent" property of either a contract or a commitment.

In an actual implementation of a REA framework, the data types and relations defined in the REA ontology should be part of the initial environment, and extensions for concrete solutions may then be added, as we will show below. The choice of tools used for declaring the model is not essential, and several different representations may coexist within a system. To illustrate examples throughout this paper we will use an XML-based notation. For sake of abbreviation we will not show all values, but abbreviate i.e. the time of occurrence and time of recording to "...".

In the case of Andy's Auto's having an ownership relation between agents and resources is no longer sufficient when car rentals become part of the business. Agent-resource relations like *rights of use* (changed at a point in time specified by a contract) and physical *control* (changed when the car is returned) may be required. These are associations of a resource that were not part of Andy's original system and both the relation type and instances of it have to be added to the system in order to implement car rentals. As an example of how a new association of an entity can be added to an existing system, we will extend the system with a *control* relation and the events affecting as follows:

```

<relationtype name="control">
  <agent> controlledBy </agent>
  <resource> controls </resource>
</relationtype>

```

The two types of events affecting the control relation are *rental* and *return*. They may be declared by adding new event types and stating what objects and relations are affected:

```

<eventtype name="rental" eType="change" relation="control">
  <time occ="..." rec="..." />
  <objId oType="resource" />
  <objId oType="agent" />
</eventtype>

```

```

<eventtype name="return" eType="change" relation="control">
  <time occ="..." rec="..." />
  <objId oType="resource" />
  <objId oType="agent" />
</eventtype>

```

The declarations of the two event types state a name for the event, that it is an alteration of a relation (as opposed to an addition where `eType="new"`), and that the relation affected is called "control". Besides from timestamps the resource affected must be stated and the new agent to associate the control relation with specified. The agent losing control of the resource does not need to be recorded, as the event-history may provide the information of who had control of the resource previously if required. Note that the two event types seem to be identical besides from their name. In the event-log they are both merely changes of control to a new agent, but when validation is performed by a rule engine there may be different business rules to observe when the two types of events occur. A rental may only be performed on a car that is grouped as a rental car, is under control of the rental shop, and has been inspected since the last rental. Also there may be rules like "a valid drivers license has been shown" to observe.

Once the event types have been declared actual instances of events may be recorded as follows:

```
<event eId=17 name="rental">
  <time occ="..." rec="..."/>
  <objId>Resource513Ford </objId>
  <objId >Agent006Bill</objId>
</event>
```

An event instance has a unique event id, an event type, timestamps and instances of the objects required by the given event type - in this case a resource (the car rented) and an agent (the customer renting it).

Note that this way of declaring events gives the flexibility of allowing records of events not typically mentioned in the context of REA-modeling, but significant for operationalizing the model. Examples might be hiring agents and adding contracts.

Recording Knowledge Level Information

Entity types may be used at the Operational Level, as part of an identification of resources without a unique id, or by commitments specifying instances of participating agent types or resource types transferred, instead of specific agents and resources. At the Knowledge Level of a business model type images can be used for stating various kinds of policies or business rules by declaring relations between entity types.

Additionally groupings or classifications may be stated at the Knowledge Level as described in [Geerts & McCarthy, 2003] and [Geerts & McCarthy, 2006]. A classification of an entity will typically state which, of a set of predefined "groupings" for the given entity type, it belongs to. Such a limited set of predefined values may from an implementation perspective be seen as an enumeration.

As an example, Andy from Andy's Auto's might discover he needs to distinguish between cars that are for sale, for rent or need to be repaired. These terms do not represent different types of cars, like i.e. "Sedan", "Jeep" and "Van" might, but different classifications of a car, and they may be changed occasionally.

Adding an enumeration and a classification event applying it, can be done as follows:

```

<enumtype name="CarIsFor">
  <option isDefault="true"> Sale </option>
  <option> Rent </option>
  <option> Repair </option>
</enumtype>

```

It is declared that the possible values of the "CarIsFor" classification are "Sale", "Rent" and "Repair" with "Sale" as the default value. Classification of a specific car may be done as:

```

<event eId="934" eType="change" class="CarIsFor">
  <time occ="..." rec="..." />
  <objId> Car0017Volvo </objId>
  <value> Rent </value>
</event>

```

Adding Dynamic Properties

Besides from intrinsic properties like i.e. the brand and type of a car, entities can have properties reflecting state changes of interest when running a business in a particular way. As the business evolves these may change and require changing the structure of existing data. Using principles of AOMs Property pattern, and the property-driven modeling approach described in [Jaquet, 2006], we may add types and instances of such properties dynamically, using declarations similar to the above. A property has a name and a value type, just like the other fields of an entity. For an example like the "ready for rental or not" state of a rental-car there are several different values of possible interest. What to model depends on the specific case, and on how conclusions are to be made. I.e. information of interest may be "when was the car last inspected?". If it was later than the last return event, then the car is ready for rental. Or, it may be "Who inspected the car?". These data can be derived by examining the event-history of inspection events on a particular car. What might also be of interest, but not available from the event-history, could be a comment on the result of the inspection "Did the car pass?". Storing the comment as a property is a possibility, but it would not seem feasible to base conclusions of a cars availability on informal comments. The properties that should be available, are those reflecting significant state changes of the car, thus we may choose one result of an inspection to be updating an *inspected* property with a boolean value. Such a property type can be declared as follows:

```

<proptype name="inspected">
  <value> boolean </value>
</proptype>

```

Note that if the conclusions depending on a property do not change, the same property type may be reused for different entities. I.e. a *color* property of a car can be changed by a corresponding *repaint* event, but if Andy decides to use his painting facilities for bicycles or dustbins the same property type *color* can be applied to these resources as well. The validation of what properties can be applied to what resources is based on policies and not handled by the event-history but by the rule engine like all other business rules. Properties are changed by transformation events described by what property they change. The *inspection* event is an example, and declaring and instantiating it may look like:

```

<eventtype name="inspection" eType="change" property="inspected">
  <time occ="..." rec="..." />
  <objId oType="resource"/>
  <value type="boolean"/>
</eventtype>

<event eId="744" name="inspection">
  <time occ="..." rec="..." />
  <objId> Car0017Volvo </objId>
  <value> true </value>
</event>

```

It is beyond the scope of this paper to provide a detailed description of our approach to implementing an event-based framework for the REA model. But hopefully the examples above have given an idea of the flexibility and adaptability provided by the meta-model, as well as an overview of the context it can be used in. Note that the examples above have been declarations of extensions to the classic REA model. The basic entities, types and relations of the REA model may however also be declared in the exact same manner allowing the initial model and its extensions to be declared uniformly. An implementation of the framework would in practice always include an initial declaration of the basic concepts.

V. Further Work

The work presented here mainly describes the task of designing an adaptable framework for REA models at a theoretical level, with brief descriptions of the different models involved and few implementation specific issues mentioned. Further work includes completing a running version of the framework suggested, thereby addressing tasks of ensuring consistency between the model representations used by the user interface and conclusions/rule engine, and the representation of the event-history we describe here. Designing tools for updating the model is also a non-trivial task.

There are, however, also interesting issues left to consider at the modeling level. The examples of event declarations given above, are in no way exhaustive and many types of declarations need to be added before the full REA model is implemented. To mention a few, commitments, contracts and several relations of the REA ontology are yet to be described before an operational version of the model is achieved. Deciding on the exact semantics of the remainder of the REA ontology is not an easy task as the on-going ontological discussions within the REA community show. However, once a decision is made on the semantics, declaring them in the same manner as the examples given is straightforward, resulting in a full declaration of the basic REA model.

Creating libraries of extensions to the basic model for use within specific enterprise types or domains, would be useful and may be based on existing research contributions. Examples may be some of the standards (ISO, UN/CEFACT, etc.) designed for a certain context, like electronic B2B transactions. Also domain specific extensions as those suggested for Supply Chain Management systems in [Hessellund, 2006], or business patterns in general as described in [Hruby et al., 2006], can be declared as reusable libraries and integrated with surrounding systems.

The declarations mentioned in this paper concern the state and relations of data and updates to these. The conclusions engine needs to share a model of the data structure with the event-history, but it also needs to know which conclusions to perform and how. The connection between the structure of entities and their behavior must also be described. AOM suggests using the Strategy pattern and RuleObjects to describe behavioral algorithms and business rules associated with entities. Exploring this approach will be part of future work.

VI. Discussion & Conclusions

This paper presents a semantic framework for an event-driven operationalization of the REA model, addressing the issues of creating and maintaining enterprise systems, that are flexible and adaptable yet remain consistent. A model describing the interaction between different parts of an event-driven framework in general is described, addressing in particular the history of recorded events that form the entry point of data to the system. A generic object-model describing metadata of the underlying data structure is suggested based on the Active Object-Model, and an example of how events can be recorded in a specific REA implementation of an event-driven enterprise system is presented.

Describing an operational instance of a REA model introduces the need to include events that are not described by the ontology, but still of a fundamental nature. Thus the event concept presented here differs from the economic events traditionally mentioned in the context of REA models. The main additions are events that add all other entities and relations of the REA, than resources and their stock-flow and participation relations, as well as events that add or modify loosely coupled properties. The AOM describes a meta model of entities, properties and their types. We adopt this structure and extend it with generic event types.

The approach of explicitly declaring relations and properties of REA entities gives an intuitive and coherent way of defining event types in a system, as they are characterized by adding or changing either a relation or a property of a specific type.

Letting all eventtypes share the same simple meta-model gives great flexibility when it comes to updating systems dynamically, as well as the security of maintaining consistent versions of the model. Adaptability and interoperability with other systems can be achieved through aligning the data exchanged, simply by adding the required properties and relations along with events to modify them.

There is however a price, for gaining the flexibility of letting the model of the framework be a simple generic meta model, allowing full flexibility of declaring entities and relations of the business model. The meta model no longer reflects the semantic restrictions and control of business model consistency, enforced by the structure of the REA model and its axioms. These semantic validation criteria are a major strength of the REA ontology, and must still be enforced, but the responsibility of validating models is moved away from the representation of the model, and into the tools for

modifying the model. They will now have to be part of the implementation logic rather than the model itself. On the other hand, such a solution allowing validation rules to be declared will be required anyway, if axioms are to be expressed for extensions to the model.

Using the property-driven approach of having loosely coupled properties, gives the possibility of reusing property types and the conclusions performed on them, with several different entity types. Achieving the full benefits of reusing properties and events, will however require some care taken when designing, documenting and managing property types, as is the case with all types of software libraries.

We believe that the proposed design of a framework would be a feasible approach for a REA-based framework in particular, but also for systems in general that are event-driven by nature and require a close mapping to real-world events, as well as a high degree of evolvability and adaptability.

References

- [Black & Black, 1929] John D. Black & Albert G. Black. *Production Organization*. Henry Holt and Company, xi edition, 1929.
- [David et al., 2000] Julie Smith David, Gregory J. Gerard, & William E. McCarthy. *Design Science: Building the Future of AIS*. <http://www.msu.edu/user/mccarth4/designsc.doc>. Referenced February 2007, August 2000.
- [Denna et al., 1993] E.L. Denna, J.O. Cherrington, D.P. Andros, & A.S. Hollander. *Event-driven Business Solutions: Today's revolution in business and information technology*. Business One, Irving, 1993.
- [Foote & Yoder, 1998] B. Foote & J. Yoder. *Metadata and Active Object-Models*. In *Proceedings of Plop98*. Washington University Department of Computer Science, 1998.
- [Fowler, 1997] Martin Fowler. *Analysis Patterns: Reusable Object Models*. Object Technology Series. Addison-Wesley, Reading, Massachusetts, 1997.
- [Geerts & McCarthy, 1999] Guido Geerts & William E. McCarthy. *An Accounting Object Infrastructure For Knowledge-Based Enterprise Models*. IEEE Intelligent Systems and Their

- Applications, pages 89–94, July–August 1999. <http://www.msu.edu/user/mccarth4/x4xop.lo.pdf>.
- [Geerts & McCarthy, 2000] Guido L. Geerts & William E. McCarthy. *The Ontological Foundation of REA Enterprise Information Systems*. <http://www.msu.edu/user/mccarth4/Alabama.doc>, August 2000.
- [Geerts & McCarthy, 2003] Guido L. Geerts & William E. McCarthy. *Type-Level Specifications in REA Enterprise Information Systems*. Working Paper, Michigan State University, <http://www.msu.edu/user/mccarth4/UTS-seminar/Type%20paper%20final%20submission.doc>, 2003.
- [Geerts & McCarthy, 2006] Guido L. Geerts & William E. McCarthy. *Policy-Level Specifications in REA Enterprise Information Systems*. *Journal of Information Systems*, 20(2):37–63, 2006.
- [Hessellund, 2006] Anders Hessellund. *Supply Chain Modeling with REA*. TR 2006-80, The IT University of Copenhagen, Denmark, 2006.
- [Hoppe, 2006] Gregor Hoppe. *Programming Without a Call Stack - Event-driven Architectures*. <http://www.enterpriseintegrationpatterns.com/docs/EDA.pdf>. Referenced February 2007, 2006.
- [Hruby et al., 2006] Pavel Hruby, Jesper Kiehn, & Christian Vibe Scheller. *Model-Driven Design Using Business Patterns*. Springer Verlag, 2006.
- [Jackson, 1983] M. A Jackson. *System development (Prentice-Hall International series in computer science)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1983.
- [Jaquet, 2006] Mette Jaquet. *A Property Driven Approach towards Describing Semantics of REA Entities*. In Mette Jaquet, Anders Hessellund, Pavel Hruby, Jesper Kiehn, & William E. McCarthy, editors, *Proceedings of the 2nd International REA Technology Workshop*, Santorini, Greece, June 2006. IT University Technical Report Series.
- [Jespersen, 2006] Jacob Winther Jespersen. *Articulating Objects*. PhD thesis, IT-University of Copenhagen, Software Development Group, 2006.

- [Johnson & Wolf, 1998] Ralph E. Johnson & Bobby Wolf. "Type Object". In *Pattern Languages of Program Design 3*. Addison Wesley, 1998.
- [Madsen et al., 1993] Ole Lehrmann Madsen, Birger Møller-Pedersen, & Kristen Nygaard. *Object-Oriented Programming in the Beta Programming Language*. Addison-Wesley, 1993.
- [Nakamura & Johnson, 1998] H. Nakamura & R. Johnson. *Adaptive framework for the REA accounting model*, 1998.
- [Sowa, 2000] John Sowa, editor. *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Brooks Cole Publishing Co, 2000.
- [Yoder et al., 2001] Joseph W. Yoder, Federico Balaguer, & Ralph Johnson. *Architecture and Design of Adaptive Object-Models*. j-SIGPLAN, 36(12):50–60, December 2001.

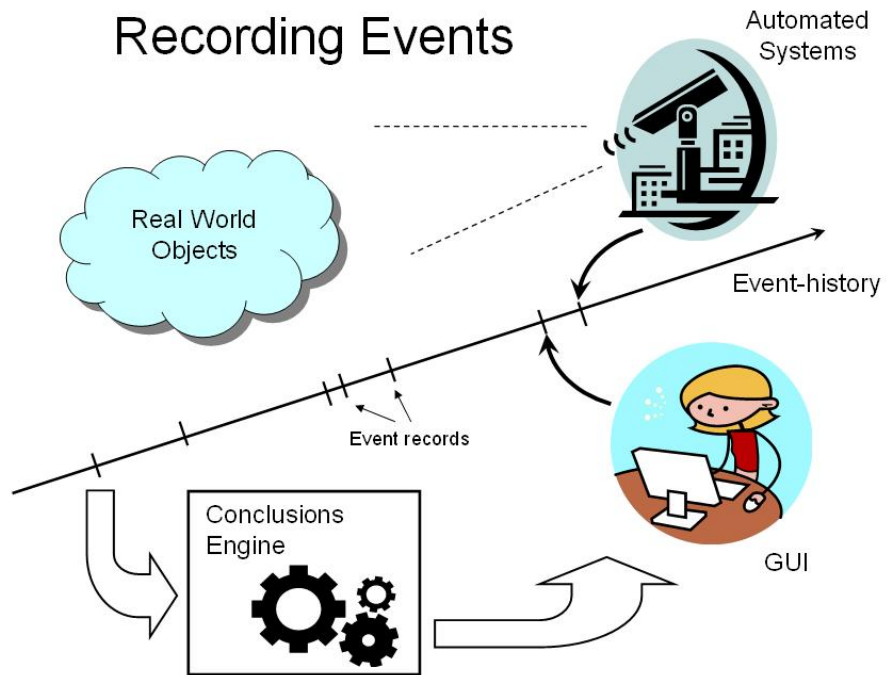


Figure 2: Creating and querying an event-history of observations

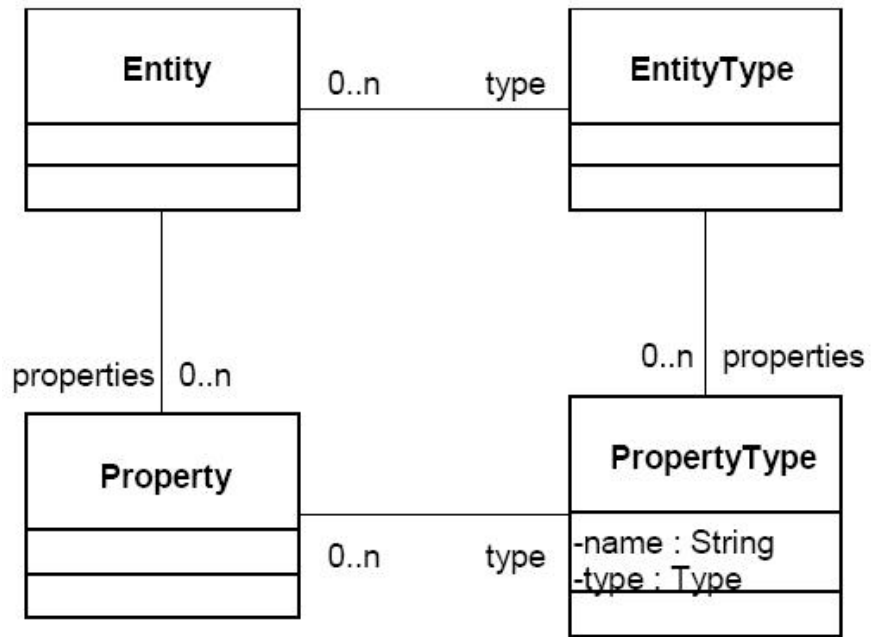


Figure 3: TypeSquare of the Active Object-Model as shown in Figure 3 of [Yoder et al., 2001]