GENERATION OF ERP SYSTEMS FROM REA SPECIFICATIONS

Nicholas Poul Schultz-Møller
Department of Computing, Imperial College London, 180 Queen's Gate, London, U.K.

Christian Hølmer
Upsido, Kgs. Lyngby, Denmark

Michael R. Hansen
Department of Informatics and Mathematical Modelling, Technical University of Denmark, Kgs. Lyngby, Denmark

Keywords: Enterprise Resource Planning systems, REA, domain specific language, Web-applications.

Abstract: We present an approach to the construction of *Enterprise Resource Planning* (*ERP*) Systems, which is based on the *Resources, Events and Agents* (*REA*) ontology. Though this framework deals with processes involving exchange and flow of resources, the conceptual models have high-level graphical representations describing what the major entities are rather than how they engage in computations.
We show how to develop a declarative, domain-specific language on the basis of REA, and for this language we have developed a tool which automatically can generate running web-applications. A main contribution is a proof-of-concept result showing that business-domain experts can, using a declarative, REA-based domain-specific language, generate their own applications without worrying about implementation details.
In order to have a well-defined domain-specific language, a formal model of REA has been developed using the specification language Object-Z. This formalization led to clarifications as well as the introduction of new concepts. The compiler for our language is written in Objective CAML and as implementation platform we used Ruby on Rails. The aim of this paper is to give an overview of whole construction of a running application on the basis of a REA specification.

1 INTRODUCTION

In this paper we present an approach to the construction of *Enterprise Resource Planning* (*ERP*) Systems, which is based on the *Resources, Events and Agents* (*REA*) ontology (Geerts and McCarthy, 2000; Hruby, 2006; McCarthy, 1982). REA is a domain-specific conceptual framework which has its use, for example, in the design of accounting and enterprise information systems. Though this framework deals with processes involving exchange and flow of resources, the conceptual models have high-level graphical representations describing what the major entities are rather than how they engage in computations.

We show how to develop a declarative, domain-specific language on the basis of the REA ontology, and together with a compilation tool, how to generate running web-applications. Hence, a main contribution is a proof-of-concept result showing that business-domain experts can, in principle, using a REA-based domain-specific language, generate their own applications without, in principle, worrying about any implementation details. This has a clear advantage compared to current technologies where the domain-specific languages have a much more imperative flavor (e.g. Microsoft Dynamics' C/AL language (Studebaker, 2007)).

The REA ontology is not formalized and the informal explanations of REA are in many ways underspecified. In order to have a well-defined domain-specific language and an associated tool we develop a formal model of REA using the specification language Object-Z (Smith, 2000; Spivey, 2001). This object-oriented and formal specification language proved well-suited for our purpose as the underlying computational model should describe the changing state of resources in ERP systems. The formalization process led to clarifications as well as the introduction of new concepts.

The formal model exhibits in a succinct way fun-

Poul Schultz-Møller N., Hølmer C. and R. Hansen M. (2008).
GENERATION OF ERP SYSTEMS FROM REA SPECIFICATIONS.
In *Proceedings of the Third International Conference on Software and Data Technologies - ISDM/ABF*, pages 12-19
DOI: 10.5220/0001872600120019
Copyright © SciTePress

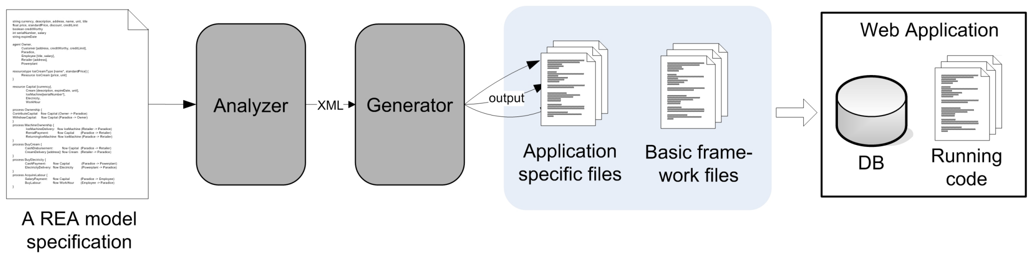


Figure 1: From REA model specification to web application.

damental properties of resources, event and agents, which any implementation of a REA-based system should satisfy. In order to achieve this, the compiler for our language is written in Objective CAML (Smith, 2006), which is a functional programming language with object-oriented features. The Object-Z model of REA could be represented in Objective CAML in a direct manner, and the functional part of the programming language proved very useful in connection with handling the properties of the model and for the compiler construction as such. As implementation platform we used Ruby on Rails (Thomas and Hansson, 2006).

The aim of this paper is to give an overview of the whole construction of a running application on the basis of a REA specification, and to justify the thesis that a running application can be automatically generated from a declarative, REA-based specification. We will sketch the flavor of the Object-Z model and how it relates to the Objective CAML implementation, to emphasize the adequacy of this approach. Furthermore, we will hint at the use of the implementation platform Ruby on Rails.

Current ERP systems try to satisfy the majority of their customers' requirements and customize the last 15-20% by using various domain specific languages such as C/AL. These languages are usually imperative and are used for various purposes such as implementation of windows, dialogs, algorithms, etc. As a result many resources are spent on customization and the extensions are tightly coupled to the basic ERP system. This prohibits the existing systems from benefitting from new advantages in language technologies and architectural designs.

We suggest another approach for creating customized ERP systems. By declaring the customer's business processes and then generate the ERP system, it will be possible to consistently extend the functionality as business processes develop. Furthermore, the approach decouples the design and implementation from the business processes. This enables transparent commissioning of new technologies.

This paper, which is based on (Schultz-Møller and Hølmer, 2007), is organized as follows: In the next section we give a brief introduction to REA on the basis of a simple example. Thereafter, in Section 3, we introduce a formal Object-Z model for REA. In Section 4 and 5 we describe the domain specific language for REA, how it is analyzed and how web-applications are generated automatically. The overall idea is illustrated on Figure 1. The last section contains a discussion of our work. In the paper we will sketch the main ideas only, for further details we refer to (Schultz-Møller and Hølmer, 2007).

2 AN INTRODUCTION TO REA

The ontology *Resources, Events and Agents* (*REA*) (Geerts and McCarthy, 2000; Hruby, 2006; McCarthy, 1982) can be used for describing business processes, such as the *exchange* of resources between economic agents, and the *conversion* (or *transformations*) of resources. REA is an ontological framework where the models typically are presented graphically like, for example, in Figure 2.

An example of an exchange process for a fictive company "Paradice", that produces and sells ice cream, is shown on Figure 2. To produce ice cream it is necessary to purchase cream from a retailer. This figure describes that the resource Cream is exchanged in a *duality* for Capital in the events Cream Delivery and Cash Disbursement, respectively.

As can be seen from the relationships, the agent Paradice *provides* the capital and *receives* cream. The *inflow* and *outflow* relationships between an event and a resource describe whether or not *value* flows into (an *increment event*) or out of the enterprise (Paradise) (a *decrement event*). Here the enterprise's capital resources decrease their value, while its cream resources increase their value – simply because the amount of each resource changes. Thus, Cash Disbursement is a decrement event and Cream Delivery is an increment event. The figure also contains an *optional* Discount event modelling that a retailer might

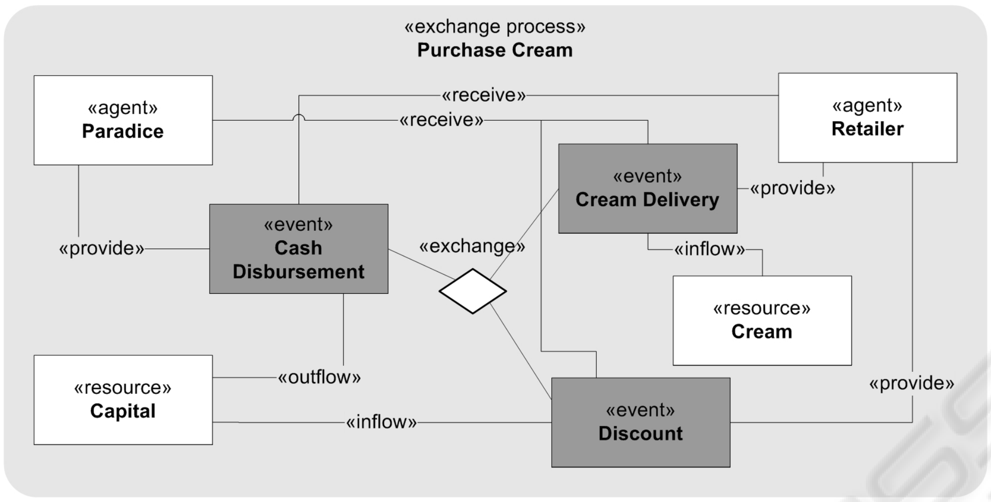


Figure 2: Modelling the purchase of cream in REA.

give a discount on a purchase. Notice that it is not possible to infer from the diagram that the discount event is optional.

Using REA, one can model a business process by defining distinct events involving agents and resources. REA has several rules that must apply to a model for it to be valid, for example: (1) Any resource must be part of at least one increment and one decrement event, and (2) Any increment event must be in a duality relationship with at least one decrement event and vice versa. Rule (1) ensures that no resource "piles up" or is sold but never purchased, and rule (2) ensures that it is always possible to track what other resources a specific resource was exchanged for.

The REA ontology does, however, not have a precise definition in a formal sense. This is a problem when a system is implemented on the basis of an ontology, as the developer may take wrong decisions which can lead to inconsistent design.

Example problems in REA are: (1) It is unclear how to distinguish two resources of the same type from each other as distinguishable resources have no notion of identity and (2) It is not possible to distinguish events that must occur in a business process and events which are optional (as the Discount event). These and other problems which have been considered and solved in (Schultz-Møller and Hølmer, 2007). We will elaborate on that in the next section.

3 AN OBJECT-Z MODEL

In this section we will sketch a formal model for REA. For the full detail we refer to (Schultz-Møller and Hølmer, 2007). The object-oriented specification language Object-Z (Smith, 2000; Spivey, 2001) was chosen for the formalization as business processes to a large extend concern the changing states and exchanges of resources. The model is divided in two parts: A *meta-model* defining the concepts of the REA ontology and their properties, and a *runtime model* defining the dynamic behavior of applications based on REA-model specifications.

We will not give a special introduction to Object-Z, but we give brief informal explanations to the shown specifications.

3.1 The REA Meta-Model

In the Meta-Model we define an Object-Z class for each entity in the REA ontology, and we formalize the rules all instances of these classes must obey. We first define *free types* in the meta-model for names and field types:

$$NAME ::= nil \mid charseq\langle\langle \mathrm{seq}_1\, CHAR\rangle\rangle$$
$$FTYPE ::= sType \mid iType \mid fType \mid bType$$

where *NAME* is either undefined (`nil` or a sequence of characters, and *FTYPE*, a field type, is one of the following **s**tring, **i**nteger, **f**loat and **B**oolean. Fields are used for augmenting the entities – e.g. an address field on a Retailer agent entity. Furthermore, fields are modelled as partial functions from names to field types:

$$FIELDS == NAME \nrightarrow FTYPE$$

The entities in the meta-model share many properties and these are specified in the *BasicEntity* class:

$BasicEntity$

$name : NAME, fields : FIELDS$

$INIT$

$name = nil, fields = \varnothing$

$InitData$

$\Delta(name, fields)$

$name? : NAME, fields? : FIELDS$

$name = nil \land name? \neq nil$

$name' = name? \land fields' = fields?$

The *precondition* of *InitData* ensures that an entity can only set data once (and in that respect works as a constructor in an object-oriented language) and that each entity is named.

A class for resources is defined by extending *BasicEntity*:

$Resource$

$\upharpoonright (name, fields, atomic, InitData)$

$BasicEntity$

$atomic : \mathbb{B}, key : NAME$

$atomic \Leftrightarrow key \in (\mathrm{dom}\, fields)$

$INIT$

$atomic = false, key = nil$

$\ldots$

where the variable *atomic* specifies whether a real world resource object is distinguishable from another of the same type. If so, there must be a way to identify a resource uniquely in the Enterprise Information System. This solution to the problem of distinguishing two resources from each other was chosen to adapt the key concept from databases.

A class for agents is defined as follows:

$Agent$

$\upharpoonright (name, fields, InitData)$

$BasicEntity$

For the definition of the *Event* class we introduce two additional free types *STOCKFLOW* (the relationship between an event and a resource) and *MULTIPLICITY* (the number of occurrences of an event in a process):

$$\begin{aligned} STOCKFLOW &::= none \mid flow \\ &\quad \mid produce \mid use \mid consume \\ MULTIPLICITY &::= \langle\langle \mathbb{N}_0 \rangle\rangle \mid infty \end{aligned}$$

Observe that inflow and outflow have been generalized to *flow*, as the direction can be inferred from the provide/receive relationship. E.g., an agent providing in an event is an outflow of a resource under possession of that agent.

A class for events is defined in Figure 3, where we, for example, observe that an exchange event can have flow only as stock flow type. Several *Events* constitute a process as sketched in Figure 4, with the most interesting invariant being [a], which states that one of the agents, called the *enterprise-agent*, must be receiver or provider in all events. The enterprise-agent is a new central concept that represents the enterprise described in a model. As will be apparent below, our system can automatically infer[1] which agent in a consistent model is the enterprise-agent.

All instances in a REA model are collected in the *MetaModel* class shown in Figure 5. The formulas [b] and [c] state that all agents and resources must be part of at least one event. Formula [d] expresses that the

$Event$

$\upharpoonright (name, fields, provider, receiver, stockFlowType,$
$\quad resource, minOccurs, maxOccurs, InitData,$
$\quad IsExchange, IsConversion)$

$BasicEntity$

$provider, receiver : Entity \cup Agent$

$stockFlowType : STOCKFLOW$

$resource : Entity \cup Resource$

$minOccurs, maxOccurs : MULTIPLICITY$

$\neg(maxOccurs = 0) \land minOccurs \neq infty$

$maxOccurs \in \mathbb{N} \Rightarrow minOccurs \leq maxOccurs$

$INIT$

$provider, receiver = Entity.INIT$

$stockFlowType = none$

$resource = Entity.INIT$

$minOccurs, maxOccurs = 0$

$\ldots$

$IsExchange$

$stockFlowType = flow$

$IsConversion$

$stockFlowType \in \{use, consume, produce\}$

Figure 3: The Object-Z class *Event*.

[1]Very unrealistic business models can be constructed which prohibit automatic inference. In these cases the user is prompted for selection of one agent.

Process

$\upharpoonright (name, dualities, INIT, InitData)$

$name : NAME, dualities : \mathbb{F}\,Event$

$\exists e_1 : dualities \bullet e_1.IsExchange \Rightarrow$ [a]
$(\forall e_2 : dualities \bullet e_1.provider = e_2.provider \lor$
$e_1.provider = e_2.receiver) \lor$
$(\forall e_2 : dualities \bullet e_1.receiver = e_2.provider \lor$
$e_1.receiver = e_2.receiver)$
...

INIT
$name = nil, dualities = \varnothing$

...

Figure 4: The Object-Z class *Process*.

enterprise-agent must be either provider or receiver in any exchange events. It is, therefore, possible to infer the enterprise-agent in a REA model. Formula [e] is central to the model presented in (Schultz-Møller and Hølmer, 2007). In the meta-model a flow of resources is interpreted as resources that either leave or enter the enterprise – not as a flow of value. Thus, any resource must both leave the enterprise and enter the system in a decrement and an increment event, respectively.

3.2 The Runtime Model

The runtime model addresses the dynamic aspects of a REA-model specification. We will not present details here – for that we refer to (Schultz-Møller and Hølmer, 2007). We will just mention two aspects:

- An *execution* of a REA-model specification is a (possibly infinite) sequence of *states*, where a state (among other things) records the *available* resources in the system. The runtime model defines all legal executions of a system, where, for example, a decrement event can happen in the current state only if there are sufficient available resources for that event.
- An agent is a person or organization, but e.g. a person may *act* as different agents in different events, e.g. as an employee in one event and as a customer in another. To cope with this, a concept *legal entity* is introduced, where a legal entity (person or organization) is associated a set of agents that the legal entity can act as in a given event.

MetaModel

$\upharpoonright (enterpriseAgent, resources, processes,$
$agents, events, InitData,$
$SpecifyEnterpriseAgent, IsValid)$

$enterpriseAgent : Entity \cup Agent$
$resources : \mathbb{F}\,Resource, agents : \mathbb{F}\,Agent$
$events : \mathbb{F}\,Event, processes : \mathbb{F}\,Process$
...

$\{p : Process \mid p \in processes \bullet p.dualities\}$
partitions $events$
$\forall a : agents; \exists e : events \bullet$
$e.receiver = a \lor e.provider = a$ [b]
$\forall r : resources; \exists e : events \bullet e.resource = r$
[c]
$\forall e : events \bullet e.IsExchange \Rightarrow$ [d]
$(e.receiver = enterpriseAgent$ xor
$e.provider = enterpriseAgent)$
$\forall r : resources; \exists e_1, e_2 : events \bullet$ [e]
$(e_1.resource = r \land$
$(e_1.stockFlowType = consume \lor$
$(e_1.stockFlowType = flow \land$
$e_1.provider = enterpriseAgent))) \land$
$(e_2.resource = r \land$
$(e_2.stockFlowType = produce \lor$
$(e_2.stockFlowType = flow \land$
$e_1.receiver = enterpriseAgent)))$
...

INIT
$enterpriseAgent = Entity.INIT$
$resources = \varnothing, agents = \varnothing$
$events = \varnothing, processes = \varnothing$
...

...

Figure 5: The Object-Z class *MetaModel*.

4 A DOMAIN SPECIFIC LANGUAGE

A domain specific language for specifying REA models has been designed so that business domain experts can express business processes. The concepts of Meta-Model are directly reflected in the language. We just give part of an example here. Consider Figure 6, which gives the specification corresponding to Figure 2.

```
string currency, description, address, unit, expireDate
agent Paradise, Retailer [address]
resource Capital [currency], Cream [description, expireDate, unit]
process BuyCream { CashDisbursement: 1 flow Capital(Paradice -> Retailer)
                   CreamDelivery:    1 flow Cream  (Retailer -> Paradice)
                   Discount:      0..1 flow Capital(Retailer -> Paradice)
                 }
```

Figure 6: Specification corresponding to Figure 2.

The types of fields are declared in the first line. Two agents, Paradice and Retailer, are declared in the second line and two resources, Capital and Cream with their fields, are declared in line 3. Then the process, BuyCream, is declared. It consists of three events. The declaration of the first event, CashDisbursement, can be read as: "CashDisbursement must occur exactly once in a BuyCream process and is a flow of Capital from Paradice to a Retailer." This example also shows an optional Discount event, where the multiplicity is 0..1. The full specification has six other processes: OwnerShip, MachineOwnerShip, BuyElectricity, AquireLabour, MakeIcecream and SellIcecream. These are omitted here for brevity as they have a similar structure.

4.1 Language Implementation

The language is implemented using Objective Caml (Smith, 2006; May, 2006), which is a programming language supporting functional as well as object-oriented programming. This makes it particularly suitable for constructing a programming model on the basis of a REA specification – the classes in the program are related to the specification and the classes in the Object-Z meta-model in a very direct manner.

The implementation is divided into two parts. First, the specification is analyzed to check whether it satisfied all properties of the meta-model. The functional part of Objective Caml proved very useful for this purpose as, for example, the properties in the meta-model, e.g. [b], [c], [d] and [e] in Figure 5, are programmed in a straightforward way. If the specification is consistent, then an XML representation is generated. In the second part, which is addressed in the next section, a web-application is generated on the basis of that XML representation.

5 AUTOMATED GENERATION OF APPLICATIONS

Having a consistent REA model, given now as an XML document, we can generate an application automatically. We have chosen to implement the application in the web-framework, Ruby on Rails (Thomas et al., 2004; Thomas and Hansson, 2006), which let us focus on developing the features of the application instead of Model-View-Control plumbing.

The generation of the implementation has three major steps:

- generation of a database schema,
- generation of classes for the specified entities, and
- generation of a runtime storage system.

The main idea is that each of the REA entities: *resources, events, agents, resource types, legal entities and processes* have a table with relations corresponding to those from the runtime model. We use the technique of Single Table Inheritance, so that when the database structure is generated all fields for the REA entities will be added to the table (e.g. field_description, field_serial_number). This gives us the flexibility of having just one database table for all specified types of the individual entities, but still letting them have individual properties. Especially when loading and saving it is useful to be able to treat them the same way. Note however that a different approach is needed for real-life ERP systems in order for the system to scale. Figure 7 gives an overview of the database.

We also generate a Ruby class for each of the Single Table Inheritance types (e.g. `CreamDeliveryEvent` a subclass of `Event`), which will refer to the entries in a database table with the same entity name (`CreamDeliveryEvent` in *events* table).

The generated entities constitute a running web-application and Figure 8 shows the web-interface generated from the Paradice example of which a part of the specification is shown in Figure 6.

A user can register an event, e.g. a purchase of cream, by simply clicking on the relevant business process, click on the event and enter the details as shown on Figure 8. If the resources are available in the current state of the database and the entered data are correct (e.g. type check of input fields) then the event is registered and the database will change state as described in the runtime model given in Object-Z.

The implementations of these events are derived

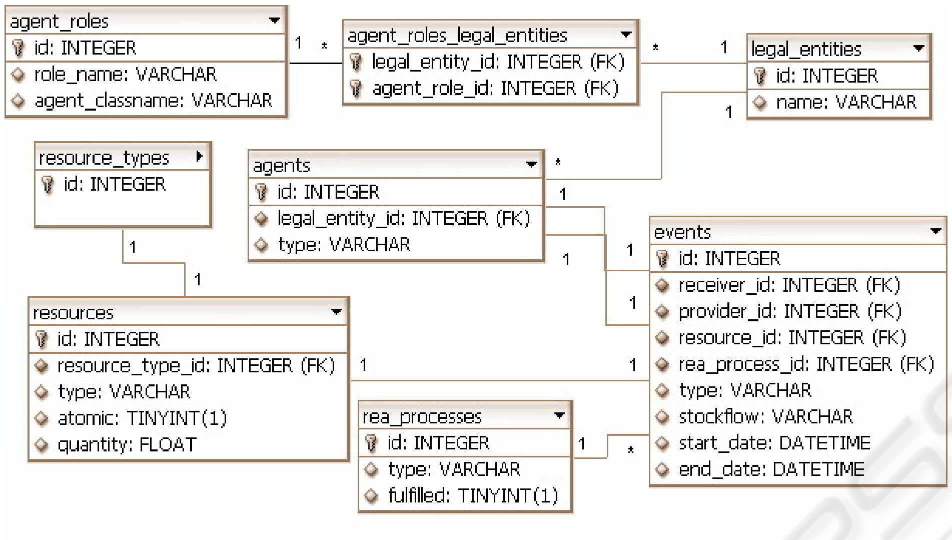
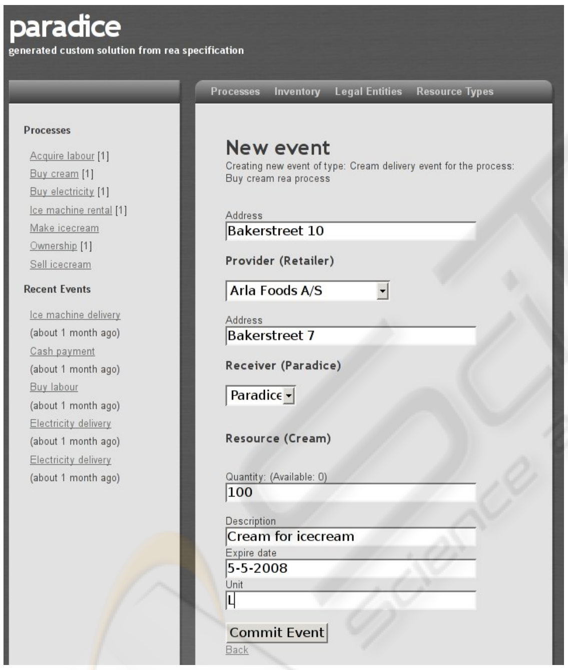


Figure 7: Database overview.


paradice
generated custom solution from rea specification

Processes Inventory Legal Entities Resource Types

Processes

Acquire labour [1]
Buy cream [1]
Buy electricity [1]
Ice machine rental [1]
Make icecream
Ownership [1]
Sell icecream

Recent Events

Ice machine delivery
(about 1 month ago)
Cash payment
(about 1 month ago)
Buy labour
(about 1 month ago)
Electricity delivery
(about 1 month ago)
Electricity delivery
(about 1 month ago)

New event
Creating new event of type: Cream delivery event for the process: Buy cream rea process

Address
Bakerstreet 10

Provider (Retailer)
Arla Foods A/S

Address
Bakerstreet 7

Receiver (Paradice)
Paradice

Resource (Cream)

Quantity: (Available: 0)
100

Description
Cream for icecream

Expire date
5-5-2008

Unit
L

Commit Event
Back

Figure 8: Screenshot of a generated application.

from the specification of the runtime model; but we will not give further details here.

Extensions to the Automatic Approach. All enterprises are ever changing and the possibility of extending a running application would be beneficial. The problem with this is that all the recorded data must stay intact and the current business processes must therefore not be changed. The simplest solution was therefore only to allow addition of new business processes and prohibit changes in the current entities in the application. In this way enterprises can adapt to changes in their business by marking old entities as outdated and replace them with new.

6 SUMMARY

We have justified the thesis of this paper, i.e. that business applications can be developed from a domain specific language based on the REA ontology. The justification is based on a prototype system which can generate web-based applications from REA specifications. So far the fundamental entities of the REA ontology, Resources, Events and Agents, have been considered and formalized, but we see no principal difficulties in extending the system to the full ontology. The prototype system is for proof-of-concept only. Currently it is not possible to express constraints on the temporal ordering of events. E.g. a system designer might want to specify that a Cash Disbursement should only take place after a Cream Delivery. However, this can also be solved in a declarative way by defining a partial ordering on events. In the specification this could be expressed as `process BuyCream` $\{\ldots\}$`[CashDisbursement` $\prec$ `CreamDelivery]`, where $e_1 \prec e_2$ means that e_2 causally depends on e_1.

In order to get a fully functional ERP system many other aspects need to be addressed, for example workflow, access control, the possibility of creating custom reports etc. Integrating these aspects into the framework is future work.

Our method is the following: first a formal model of REA is expressed in Object-Z. This clarified many vague points about REA, and forms the basis for the definition of a domain specific language for declaring business processes and its implementation. The implementation consists of an analyzer and a compiler, both written in Objective Caml. Objective Caml is a functional programming language with object oriented features, and with this language we can express the Object-Z model for REA (classes, operations and rules) in a straightforward way. The analyzer performs the consistency checks as stated in the formal model, and the compiler generates a web-based application satisfying the rules of REA. As implementation framework Ruby on Rails was used.

ACKNOWLEDGEMENTS

We are grateful for comments and suggestions from Ken Friis Larsen.

This work is partially funded by ARTIST2 (IST-004527), MoDES (Danish Research Council 2106-05-0022) and the Danish National Advanced Technology Foundation under project DaNES.

REFERENCES

Geerts, G. L. and McCarthy, W. E. (2000). The ontological foundation of rea enterprise information systems.

Hruby, P. (2006). *Model-Driven Design Using Business Patterns*. Springer.

May, J. H. (2006). *Introduction to the Objective Caml Programming Language*.

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

Schultz-Møller, N. P. and Hølmer, C. (2007). Bachelor Thesis: Tool Support for Business Processes – REAML, The REA Language and Metamodel.

Smith, G. (2000). *The Object-Z Specification Language*. Kluwer Academic Publishers, University of Queensland, Australia.

Smith, J. B. (2006). *Practical Ocaml*. APress.

Spivey, J. M. (2001). *The Z Notation, A Reference Manual*. Prentice Hall International (UK) Ltd, 2nd edition edition.

Studebaker, D. (2007). *Programming Microsoft Dynamics NAV*. Packt Publishing.

Thomas, D., Fowler, C., and Hunt, A. (2004). *Programming Ruby*. Pragmatic Bookshelf, 2nd edition edition.

Thomas, D. and Hansson, D. H. (2006). *Agile Web Development with Rails*. Pragmatic Bookshelf, 2nd edition edition.