2015 IEEE 17th Conference on Business Informatics

Model-Driven Retail Information System based on REA Business Ontology and Retail-H

Bernhard Wally, Alexandra Mazak, Bernhard Kratzwald, and Christian Huemer
Business Informatics Group, Institute of Software Technology and Interactive Systems
Vienna University of Technology, 1040 Vienna, Austria
{wally, mazak, kratzwald, huemer}@big.tuwien.ac.at
http://www.big.tuwien.ac.at/

***Abstract*—Enterprise resource planning (ERP) systems are often cumbersome to customize to a client's needs. Model-driven approaches promise to simplify these attempts. In this work we present an ERP prototype based on the *Resource-Event-Agent* (REA) business ontology that follows a "model-at-runtime" approach: one may customize the ERP system during runtime by changing the underlying REA models. We are using *Retail-H* as the reference framework for building a retail information system (RIS). Our main contribution is the prototypical implementation of a domain agnostic REA engine that can be loaded at runtime with domain specific business models—these models can further be manipulated at runtime. On that basis we have exemplarily modeled main concepts of Retail-H in REA. Validation of the implemented components is realized by applying real business activities and requirements received from our partner, a business software solution provider.**

I. Introduction

The Resource-Event-Agent (REA) [1] business ontology and its extensions has received lots of attention from the value modeling community, apart from its original accounting roots. Its application in merchandise and retail information systems (MIS/RIS), or more generally in enterprise resource planning systems (ERP), has been discussed and several implementations have been undertaken [2]–[4]. However, to the best of our knowledge, there has not been an implementation that is based on a domain-agnostic REA engine.

REAlist is the name of a prototype we have recently built in close collaboration with a business software solution provider. It is a prototypical web-based RIS application for trading companies built on top of a generic REA core system. It is implemented as a web-based (i.e. is accessed through a browser), multi-tenant-aware (i.e. multiple tenants with different business models can be served from a single server instance) software-as-a-service (i.e. customers do not install and maintain their own servers). The business domain agnostic REA core was designed to potentially drive a full ERP system, however in the REAlist prototype only a first subset of that greater endeavor was implemented: many of the core parts of a retail information system.

With the REAlist prototype we have implemented a generic REA engine. As a proof-of-concept we have fed it with data for being prepared to handle business objects of trading enterprises. For the formulation of these trading business objects, we have modeled concepts from Retail-H [5], [6] using REA elements. In REAlist this business modeling takes place at runtime, i.e. no re-compilations or database re-engineering is necessary, as all models and their instances, the business data objects, are stored in a generic database (cf. Sec. III for how this is realized in terms of a software architecture and Sec. V for how this emerges in terms of runtime objects).

While REA provides a modern and flexible concept for capturing economic events that are reflecting double entry bookkeeping (after all, it was initially established as an accounting framework), the REA entities we have developed enable a mapping from REA to "legacy" accounting artifacts and are used to feed an external bookkeeping software. Also, in REAlist we have implemented functions such as a key performance indicator service and an interactive balance list view.

II. Related Work

A. REA Business Ontology

The REA [1] business ontology resembles a high-level conceptual modeling language for the definition of business models. While it was originally developed for the accounting domain, its versatility and high level of abstraction led to applications in the business and value modeling community [7]–[10]—in 2007 it has become an ISO standard as an accounting and economic ontology for transaction scenarios in business operations [11]. In its core, REA declares a generic trading pattern: *economic events* deal with *economic resources* and are driven by *economic agents*. Events can increment the quantity on hand of a resource (e.g. receiving money from a customer) or a decrement the quantitiy on hand of a resource (e.g. giving away goods to a customer). Two or more of such opposed events are linked by a *duality* relation and form a *transfer*. In REA the transfer is the fundamental concept of economic activities. In order to complement events ("what is" or "what has been" happening in my company) with means for planning, the concept of *commitments* was introduced [8]: structurally similar to events, commitments represent parts of a contract—increment and decrement commitments are related via a *reciprocity*. For a "running example" of REA, please refer to Sec. IV.

Domain specific REA constellations are usually presented via entity relation (ER) diagrams or OMG's[1] UML[2] class diagrams. In [12] a graphical domain specific language (DSL) was developed, the *REA-DSL*. It enables the definition of

[1]Object Management Group, cf. http://www.omg.org/
[2]Unified Modeling Language, cf. http://www.omg.org/uml

978-1-4673-7340-1/15 $31.00 © 2015 IEEE
DOI 10.1109/CBI.2015.49

REA entities in a graphical editor manner and facilitates an abstracted view of the business model in the sense of a value chain (flow of resources through dualities). A similar approach has been followed by [13], where different graphical notations have been envisioned, briefly evaluated and one of them was implemented in a graphical workbench. In our approach we have designed our basic business model using REA-DSL, but we have not integrated it into the prototypical REAlist implementation, mainly for technological reasons (different underlying technologies and target systems). Instead, we are using a tree-based REA business modeler (similar to the one presented in [4]), where REA entities can be hierarchically defined and can be linked with each other, e.g. the agent hierarchy and the resource hierarchy can be separately defined and entities thereof can be used in participation or stockflow relations in commitments or events.

B. Model-Driven ERP

ERP software applications are an instance of integrated standard business software [14] that comprise information from at least three of the following resources [15]: material, employees, production, logistics and finances, and that support business functions such as purchasing, production, planning, invoicing, sales, etc. [15]. In REAlist all of the above mentioned resources can be handled, but only material, logistics and finances have been assessed in the runtime model so far. The functions supported in REAlist include purchasing, invoicing, logistics, sales and billing (cf. Sec. V).

Unlike in traditional ERP systems, where business models are used for the initial communication between the enterprise that wants to make use of an ERP system and the software provider or consulting company, model-driven ERP systems rely on business models as the primary vehicle of communication throughout the lifecycle of the system [16].

A different approach on model-driven ERP was taken in [17], where OMG's MDA[3] was employed, by making use of UML's profile mechanism. Customization of the ERP system is realized by specifying ERP components in UML syntax and then transforming such an UML model into parameters for an already existing ERP system.

In [18] it is stated that the configuration of enterprise systems is consuming significant resources, and that model-driven approaches could help reducing resource unthriftiness. For that, an extension to event driven process chains (EPCs) is proposed: configuration nodes and configuration attributes. EPCs in the reference model can be extended by configuration information and the client specific configuration realization is then based on the configurability of the reference model.

C. Retail-H

Retail-H [5], [6] is a detailed data and function framework for the trading domain that can serve as a guideline for the implementation of trading-centric RIS applications. The structural concept with the main building blocks, shaped like a house, is depicted in Fig. 1: the gray building blocks in the middle resemble different operational processes, that follow a top-down and left-to-right order. The basement of the house represents tasks from business administration, while the roof stands for evaluation and analysis systems. The complete house resembles a *retail information system*, the gray *H* corresponds to the functions and data a *merchandise information system* typically implements. In fact, many fundamental concepts of Retail-H are of use for e.g. manufacturing businesses (such as procurement of raw material and distribution of products), but Retail-H is purposefully completely missing production-related concepts—consequently in the REAlist prototype we are also not considering production. However, due to the open architecture and versatile modeling layer, production support could be added to the software on a plug-in basis.

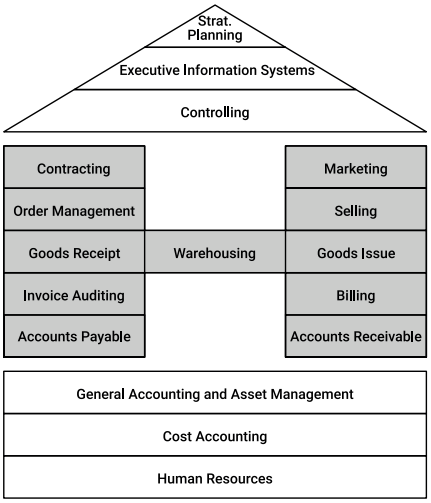

Fig. 1: Structure of Retail-H [6]—note the name-giving H-shaped area shaded in gray. The top-entry on the roof reads "Strategic Planning".

III. REALIST ARCHITECTURE

REAlist is built by realizing the software architecture depicted in Fig. 2. The type-object pattern [19] is applied in the modeling approach, i.e. REA entities are implemented *dual*: as (i) a type entity and (ii) an object entity (e.g. `agent_type` and `agent`). An object entity instance is related to exactly one type entity instance via a `typification` association. Instances of type entities are used to model the business by declaring the domain vocabulary and the relations between the elements—instances of type entities are shaping the "type layer", whereas instances of object entities are subsumed in the "object layer".

`REA DB` represents the database (DB) level: it consists of two areas, (i) the business model and (ii) the business data. In both cases, in contrast to other REA implementations, the tables reproduce the structure of the meta-model (which is the REA business ontology), i.e. tables are named `agent_type`, `event_type`, etc. for the business model, and `agent`, `event`, etc. for the business data. Due to its "model-at-runtime" nature, REAlist does not allow nor require any domain-specific tables, such as `payment_event`, etc. The business model is specified by creating and referencing rows within the type layer (i.e. among instances of type entities).

[3]Model Driven Architecture, cf. http://www.omg.org/mda/

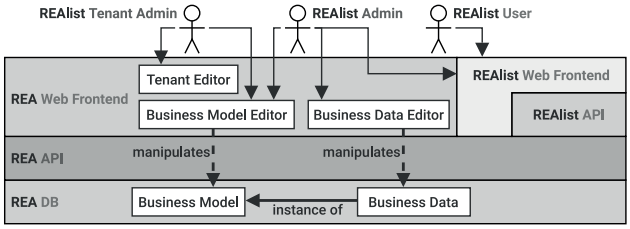

Fig. 2: Software stack of REAlist: Stack elements starting with "REA" are handling raw REA entities, while elements starting with "REAlist" handle structures and follow functions of Retail-H and RIS applications.

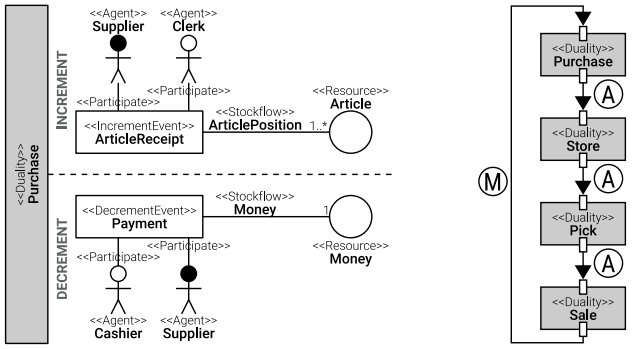

Fig. 3: Instantiated REA concepts: (left) a duality with its associated events, depicted in a slightly modified REA-DSL [12] notation, and (right) a value chain view of multiple dualities.

Business data, such as concrete agents or events is reflected in the database as rows within the object layer.

REA API is the persistence engine that abstracts the database in terms of a software API[4]. In REAlist, entity tables are abstracted as classes, which have been implemented in terms of an object oriented programming language. The API provides means for the handling of type entities and object entities.

REA Web Frontend is a web browser based graphical user interface (GUI) for the manipulation of business models, business data and REAlist tenants. The declaration and editing of the business model is intended to be done by using "raw" REA concepts, as such the generic (i.e. business domain independent) `Business Model Editor` (that operates on the type layer) is sufficient for such administrative tasks. The `Business Data Editor` that operates on the object layer has been included as a "window to REA", it is not meant as a tool for capturing business events on a daily base, but provides insight into the stored business data. The `Tenant Editor` enables handling of the multiple tenants that a single REAlist server instance can drive. Through it the `REAlist Tenant Admin` can model reference business models that are not connected to any tenant but can be used as blueprints when creating new tenants. Also, business models of existing tenants can be manipulated, and the tenants' users can be administered by defining the set and configuration of frontend plugins and the structure of the navigation menu (see `REAlist Web Frontend`).

REAlist API is the implementation of a retail information system based on Retail-H, and it consists of two parts: (i) the structural components of the RIS are defined as instances in the type layer, and (ii) the behavioral components are implemented in code. It abstracts from the REA API by providing mappings from RIS concepts to REA concepts (cf. Sec. V).

REAlist Web Frontend is a GUI that is accessible through a web browser that provides functions of RIS and hides REA concepts from users other than business model editors. It is implemented via a light-weight plug-in framework that enables (i) developers packaging the GUI into separate modules and (ii) administrators defining the navigation menu structure for web frontend users. Parts of the GUI have been presented in [20] already.

REAlist Tenant Admin is the user role for the system owner, i.e. the person/company running a REAlist server instance. It can manipulate tenants and their meta data as well as their users and business model (see above).

REAlist Admin is the user role for tenant-specific business model editors. They are internal or external experts that have prolifient knowledge of a business' structure. They have full access to the `REA Web Frontend`, allowing them to view and manipulate the business model (type layer) and the business data (object layer) by directly working on REA entities. They have also full access to the `REAlist Web Frontend`, as anything that can be done there can also be realized in the `REA Web Frontend`.

REAlist User is the user role for tenant-specific "application users" that are using the REAlist RIS on a daily base. They can edit business data by providing information following the workflow that is defined by the `REAlist Web Frontend` plugins such as issuing purchase orders or executing invoice auditing.

IV. REA by Example

Fig. 3 shows the core REA structure by an example: an increment event `ArticleReceipt` and a decrement event `Payment` are dual events of the `Purchase` duality which is a *transfer* between inside and outside parties. In this case, the resource `Article` is acquired in exchange of `Money`. The increment and the decrement event require participating agents from different companies in order to resemble a valid transfer—here the outside agent `Supplier` is providing articles (for the ordering party this is an increment event) and receiving money (for the ordering party this is a decrement event), whereas the receiving participant in the increment event is a `Clerk` and the providing participant in the decrement event is a `Cashier`. In contrast to transfer dualities there exists a second duality type, the *transformation* duality. Here, decrement events again decrement the quantity on hand of a certain resource, while increment events increment it. With the transformation duality it is possible to model manufacturing processes and other internal processes that are required for the operation of a business, such as warehousing. In the latter case the consume and produce events do not really decrement and increment the quantity on hand of resources, but a property of a resource is manipulated, in this specific case the location of the resource. The transformation duality can thus be interpreted

[4]Application Programming Interface

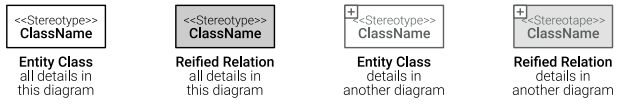

Fig. 4: Legend for subsequent class diagrams: classes with white background resemble REA entities (e.g. `Resource`, `Agent`), classes with gray background depict reified REA relations (e.g. `Duality`, `Stockflow`). Grayed out elements with a "plus" symbol in the upper left corner (the two symbols on the right) depict elements that are part of the currently described model, but are explained in detail in one of the other diagrams. For space reasons, their details and related elements are not repeated.

as "the resource is lost in location A but appears in location B".

When "zooming out" of a detailed duality view such as the one on the left side of Fig. 3, the duality can be viewed as a black box with certain resources flowing in and certain resources flowing out. This is referred to as the *value chain* view. On the right side of Fig. 3 a value chain is shown that combines multiple dualities and thus reveils the resource flow of the company: in the `Purchase` duality money Ⓜ is exchanged for articles Ⓐ, in the `Store` duality (which is a *transformation* duality) the articles are put into the warehouse, in the `Pick` transformation duality the articles are retrieved from the warehouse and finally in the `Sale` duality the articles are exchanged with customers for money which can be used for another purchase, etc.

V. REA meets Retail-H

In REAlist we have implemented parts of the Retail-H model (i) structurally by a set of REA entities, and (ii) on a behavioral level by a set of `REAlist Web Frontend` plugins that provide a certain workflow following specifications of our business partner. The behavioral part is not included in this paper, whereas the structural part is presented in the following sections, following the building block structure of the "H" of Retail-H. As Retail-H exercises a structural analogy of the left and the right side of the "H", we will only describe the left side (procurement) and the bridging block (warehousing) in detail here, but we will not explain the right side (distribution), since most of it can be inferred by mirroring procurement concepts to distribution.

In this section (*after* this paragraph) we are using an Upper-Case-First Upright Font Style to note references to elements of the REA meta-model such as Resource, Event, and Agent, and *a lower-case italic font style* to note elements of the type layer, such as *article*, *payment*, and *employee*. *[Sidenote: The decision for this rather unusual typesetting comes from readability issues we have had with previously used typesettings, including e.g. monospaced fonts, caused by the density of marked entities.]* Elements on the type layer in turn denote descriptors for elements on the object layer, hence subsequent class diagrams implement the graphical notation depicted in Fig. 4, with the following semantics. The stereotype of the class constitutes the meta-class that is to be instantiated in business data generation, and the class name denominates the name of the corresponding type entity instance: a class named `Article` of stereotype `Resource` implies (i) that a type entity of type `ResourceType` has been instantiated by the name of "Article" and that (ii) the corresponding business data will be of type `Resource`, with a typification association to the `Article` instance. In order to improve readability, labels in class diagrams adopt a "camel-case" capitalization for compound words, whereas in the textual description words are spelled out separately and transformed to lower case, e.g. `ReturningWarehouseKeeper` in the diagram will become *returning warehouse keeper* in the text.

A. Procurement

Definition 1 (from [5, translated]): *Procurement* combines purchasing, i.e. administrative tasks that provide a framework for contracting and inbound logistics, i.e. supply with wares and goods in a profitable and market compatible manner. Procurements consists of five sub-processes: contracting, order management, goods receipt, invoice auditing, and accounts payable.

In REAlist we are following this scheme and provide REA structures for the mapping of these five subtasks. The procurement as a whole can be established by merging all subtasks into a single class diagram. In order to make REAlist compliant to current financial regulations, REA structures need to be augmented with information that allows mapping of REA occurrences that affect accounting to corresponding profit and loss accounts. We are using Properties of REA entities for that purpose—these Properties are named `⋆AccountNo` in the class diagrams and * *account number* in the text (the * denoting a wildcard for domain specific information, e.g. whether that account number is used for outbound or inbound transactions).

Definition 2 (from [6]): "*Contracting* makes the basic procurement decisions and updates the relevant base data. Central tasks are determining suppliers [. . .] , goods to be obtained from these suppliers, negotiating the price and conditions framework [. . .] and possibly determining value and quantity contracts or delivery schedules."

Synopsis: Contracting (cf. Fig. 5) information that is supported in our prototype by agents that define certain master data records, and by a set of policies that represent available goods as well as the the price and conditions framework. What is not supported in the REAlist prototype are advanced (e.g. repeating) value and quantity contracts as well as delivery schedules.

The supplier's master data is implemented as a *supplier* Agent with Object Properties (*supplier number*, *supplier account number*) as well as a Dependency to an *address* Location (*street*, *zip*, *city*, *country*). The *articles* that can be purchased from a given *supplier* are defined by a set of *purchase list price* Policies, where each of these Policies defines a *price*, as well as a *start* and an *end* date, defining the validity time range of the list price. Further contracting information includes *purchase turnover bonus* Policies where an ex-post *percentage discount* is provided from the supplier if a certain *lower range* of turnover has been reached within a year and the *upper range* has not been exceeded. A *purchase list discount* Policy can

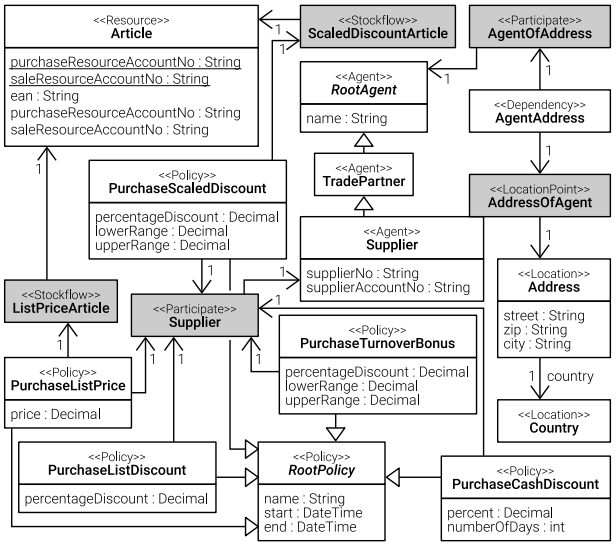

Fig. 5: Contracting information elements, modeled at runtime on the type layer.

be granted simply for listing an item. *purchase cash discount* Policies define the standard rule for cash discounts given in *percent* based on the *number of days* that have passed since the date of the incoming invoice. *purchase scaled discount* Policies enable the definition of discounts where a given *percentage discount* is granted for a specific *article* in case a certain *lower range* has been reached and the *upper range* has not been exceeded.

Definition 3 (from [6]): "*Order management* involves placing orders by determining the quadruple: supplier, article, quantity and time [. . .]."

Synopsis: Order management (cf. Fig. 6) is represented by an order contract comprising an increment commitment (supplier promises us to deliver articles) and a decrement commitment (we will pay for the articles). The order list items are encoded as stockflows that hold information for correct taxation and possible discounts or additions (e.g. shipping fees) that apply to this order. Additional agents can be specified in order to internally plan the goods receipt process. The commitments and their stockflows and participants together with the suppliers master data yield all the information that is required to e.g. generate a purchase order document.

Order management is realized by the *order* Contract that involves two Commitments: *article receipt* and *payment*. The outside Participate relation involves an Agent of type *supplier* who is the *supplier* Participant of ordered *articles* and the *payment receiver*. On the inside a *clerk* is responsible for the ordered articles, while a *cashier* is responsible for the *payment*. The *article receipt* Commitment allows reserving a number of inside Agents that should participate in the actual *article receipt* Event: as the *invoice recipient* a *cashier* can be specified who will be the addressee ("to the attention of") of the invoice. The *warehouse keeper* behind the *delivery recipient* Participate relation is the Agent that is responsible for physical reception of the articles, whereas the *delivery point*

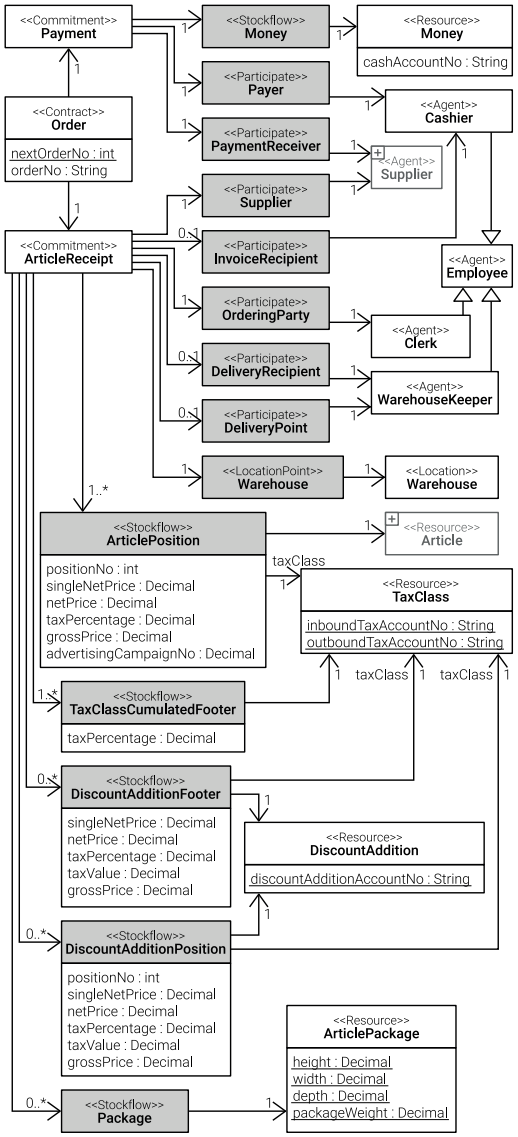

Fig. 6: REA entities relevant for order management, modeled at runtime on the type layer.

participation denotes the Agent the delivery is addressed to. Additionally, the Location Point relation *warehouse* points to the physical address the articles should be shipped to. The *start* and *end* date Properties of the *article receipt* Commitment are used to define a delivery interval, while the corresponding Properties of the *payment* Commitment denote the agreed payment interval.

Besides the *money* Stockflow of the *payment* Event that involves a *money* Resource identified by a *cash account number*, the following Stockflows are involved in the *article receipt* Commitment: *article position* represents a single order line item and relates to a specific *article*. For the current order, the pricing information is stored in each Stockflow, including the total *gross price*, the total *net price* and the *single net price* (per each) of the line item. Further, the *tax*

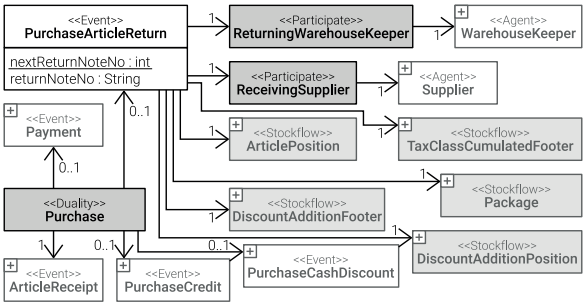

Fig. 7: REA entities relevant for goods receipt, modeled at runtime on the type layer.

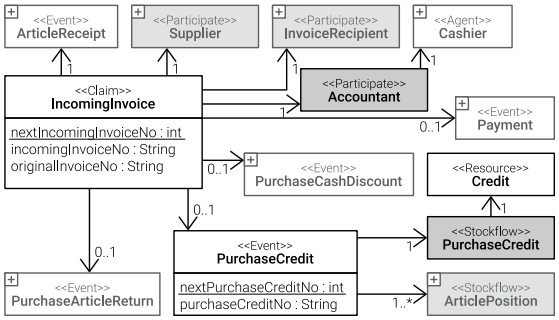

Fig. 8: REA entities relevant for invoice auditing, modeled at runtime on the type layer.

class Property specifies the *tax class* Resource this line item is related to; the taxed amount can be calculated from the *tax percentage* Property. *advertising campaign number* is an optional Property that identifies a certain advertising campaign this line item references. The *position number* provides an identifier for potential *discount addition position* Stockflows that represent discounts or additions for specific line items (obtained through e.g. *purchase scaled discount* Policies). Each *discount addition position* specifies pricing information in an analogous manner as in the *article position* Stockflow. The same Properties are declared in the *discount addition footer* Stockflow that specifies discounts or additions that are affecting not single line items but the complete order, such as shipping costs and service fees. The *discount addition* * Stockflows relate to a *discount addition* Resource which holds a Property for providing the *discount addition account number*. The optional *package* Stockflow provides, through the related *article package* Resource, information about some physical properties of the packages, such as weight and dimensions. in conjunction with the current state of the warehouse and the physical size of its shelves and bins a planning module could potentially compute instructions where goods should by physically stored once they have been received. With all that, the required quadruple is complete: a *supplier* is specified, the *articles* to be purchased and their quantities are defined, and the time frame for delivery and payment has been agreed on.

Definition 4 (from [6]): "*Goods receipt* is the [. . .] logistical equivalent to the purchasing order. [. . .] [it] also covers physical goods storage, recording the goods receipt and analyzing the delivery notes with the assessment of the goods receipt, an update of the inventory and [. . .] the handling of returns [. . .]."

Synopsis: With the goods receipt process (cf. Fig. 7) the increment part of the order contract is fulfilled. Through an increment event it is recorded which articles have been received (and kept) and through a decrement event which articles (if any) have been returned to the supplier. The articles are then stored using a transformation duality (changing the location property of the articles) and the inventory is updated. The increment event holds all the information needed to generate an article receipt slip, while the decrement event provides the information needed to generate a return note document.

Goods receipt involves the Event that fulfills the *article receipt* Commitment. It is a structural copy of this Commitment, thus its details can be viewed in Fig. 6. In the *article receipt* Event the delivery date and involved Agents are stored. What's new is the *purchase article return* Event that captures the handling of returns: the Participations comprise the *receiving supplier* and the *returning warehouse keeper*. The *purchase article return* Event reuses Stockflow definitions from *article receipt*, as the information that needs to be recorded is the same. A *return note number* is generated and stored with the Event that allows printing return notes with a traceable identifier. With the *article receipt* Event a *purchase* Duality is created and initially set to an unbalanced state—no corresponding *payment* Event has occurred yet.

Physical goods storage is depicted in Fig. 10 (cf. Sec. V-B), where a *store* transformation Duality is utilized to record information about the storage process: in the *grab* Event a *warehouse keeper* in the role of a *grabber* grabs one *article*, quantified via the *grabbed article* Stockflow fraom a *grab location* that is a *warehouse*. In the *store* Event that *article* is stored in a *store location* which is a *bin* (cf. Sec. V-B). With that, an update of the inventory is executed by creating or updating instances of *article storage area* Dependencies: the *article located* Stockflow specifies the quantity of *articles* that are stored in a certain *storage area*.

Definition 5 (from [6]): "The value-based equivalent to the goods receipt task are the *invoice* arrival and *auditing* [. . .]. [. . .] invoiced quantities must be compared with the order [. . .] , the delivery note [. . .] and the goods receipt slip [. . .]. [. . .] invoiced values must be compared with the agreed price and conditions scheme."

Synopsis: The invoice (cf. Fig. 8) is modeled in REAlist by a claim. The presence of an increment event without a corresponding number of decrement events affords the creation of a claim, which is an explicit statement that there is an imbalance. While it can be argued that an explicit claim instance is not necessary, as the increment and decrement events together with their corresponding commitments should reveil that imbalance, the claim entity makes this imbalance explicit. The claim entity can then be equipped with information such as the invoice date, invoice number, etc. With regards to the invoice auditing process, the creation of the claim states that the given event imbalance is internally accepted and that a payment process should be triggered. If articles have been returned previously, a purchase credit might have been agreed on that will reduce

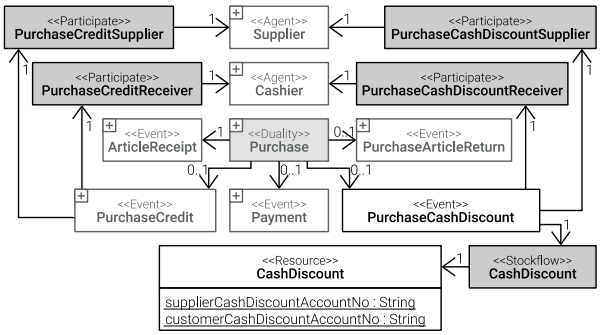

Fig. 9: REA entities relevant for accounts payable, modeled at runtime on the type layer.

the amount to pay in the current order (coming to effect in the accounts payable process).

With the reception of an incoming invoice the auditing process is triggered. The *purchase* Duality, where the goods receipt slip is implicitly defined, is taken as an initial ground truth for the auditing and compared to the delivery slip and the invoice of the *supplier*. Irregularities need to be discussed and resolved with the *supplier* and potential changes need to be made to the underlying instances, such as the quantity of *articles*, or the amount of a discount. When the auditing is positively resolved, an *incoming invoice* Claim is created. This *incoming invoice* can (for traceability reasons) relate via the *original invoice number* to the invoice the *supplier* has issued and that this Claim corresponds to. Also an internal *incoming invoice number* is created and recorded for the Claim. The materialization of that Claim is defined by an *article receipt* Event, which (for the trading company) always is the benchmark for related processes. In REAlist the *accountant* Participant (the Agent responsible for executing the invoice auditing) is of type *cashier*. In the case that *articles* have been returned to the *supplier*, a *purchase credit* Event is recorded that relates to a *credit* Resource (this recording allows extracting the correct accounting record later), as well as, through an *article position* Stockflow to the *articles* that have been returned.

Definition 6 (from [6]): "The major task of the *accounts payable* task is handling payments [. . .]. Credit notes and subsequent billings may need to be booked. [. . .] "

Synopsis: In terms of REA, payment is recorded by a set of events (cf. Fig. 9): a payment decrement event can be accompanied by an increment event recording a received cash discount.

Accounts payable completes the *purchase* Duality by recording the Events that settle the *incoming invoice* Claim created in the previous step. The most obvious Event is *payment*, where a certain amount of money is transferred from a specific *money* Resource to the *supplier*. Specific situations might require additional or different events: *purchase cash discount* is a common concept where a percentage discount is granted by the *supplier* in case the payment has been realized before a given time limit (both the percentage and the time limit ar usually negotiated in the contracting process and stored

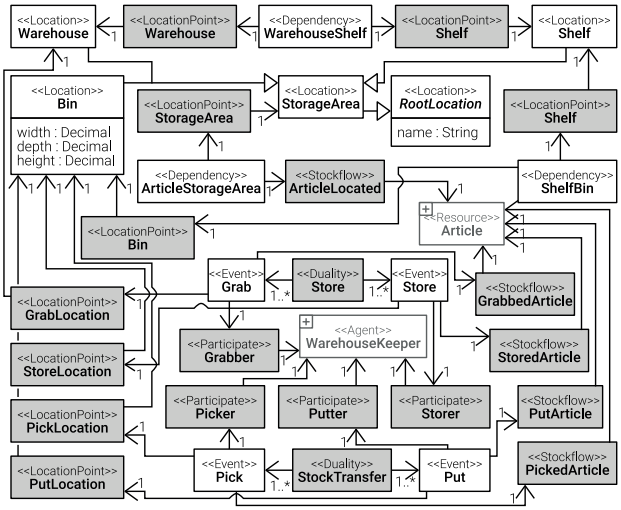

Fig. 10: REA entities relevant for warehousing (and partly also for goods receipt), modeled at runtime on the type layer.

in the corresponding Policy entities). The quantity of the cash discount is noted in the *cash discount* Stockflow and recorded on the *cash discount* Resource.

B. Warehousing

Definition 7 (from [6]): "*Warehousing* performs the bridging function between the procurement side and the sales side. [. . .] subtasks involve updating the warehouse master data, stock transfers and posting transfers, stocktaking in the warehouse or in the branch and warehouse control. [. . .] "

Synopsis: REAlist models warehouses that contain shelves that in turn contain bins (this setup does not correspond exactly to the warehouse model of Retail-H due to requirements from our business partner, but a structural change to the Retail-H model could be realized at runtime in case this model would be preferred).

Warehousing in REAlist is modeled as presented in Fig. 10: a tenant, i.e. a trading company, can be in control of multiple *warehouse* Locations. *warehouses* can be related to *shelf* Locations via *warehouse shelf* Dependencies. *shelves* in turn can be related to *bin* Locations by *shelf bin* Dependencies. This results in a setup where a *warehouse* comprises multiple *shelves* and each *shelf* comprises multiple *bins*. It is the responsibility of the application built on top of the REA model to make sure that a certain *shelf* belongs to only one *warehouse*, etc. In order to retrieve the recorded current state of a *warehouse*, the Dependencies of type *article storage area* need to be inspected as they hold the mappings of *articles* to *storage areas*.

Stock transfers can be realized by utilizing the *stock transfer* transformation Duality: a *warehouse keeper* in the role of a *picker* picks *articles* from a *pick location* (a *bin*) and puts it in the role of a *putter* to another *bin* related by the *put location*. The corresponding *article storage area* Dependencies need to be updated accordingly.

C. Distribution

Definition 8 (from [5, translated]): *Distribution* summarizes decisions and operations that affect the path of goods to a customer. In Retail-H it is structurally analogue to the procurement process.

Synopsis: In relation to this definition we have mirrored the procurement side to the distribution side of REAlist. Thus we are not showing the details here, as most of the statements would be repetitions of previous statements but "with opposite sign".

VI. Evaluation

We have put the REAlist prototype to the test by running through actual business cases that have occurred at our business partner. For that, we have rebuilt those business in REAlist cases to see, whether the data we can receive from the system conforms to the data expected. Suppliers and customers have been added to the system, together with their master data and the set of policies that has been agreed on (list prices, list discounts, cash discounts, etc.). Then, a few business cases have been inserted, by following the building blocks of Retail-H. After each step a PDF[5] file was generated: a purchase order, a goods receipt slip, a return note and an incoming invoice for procurement and a sales order, a delivery slip and an outgoind invoice for distribution. In order to check whether incoming invoices have been settled, a balance list can be generated showing unsettled claims for both suppliers and customers. The PDF files were compared to the ones from real business cases and successfully checked for congruency. Also intermediate states, e.g. after invoice auditing but before accounts payable have been compared to expected values by looking at the generates balance lists—again, the values matched.

The accounting meta-information that was added to selected REA entities via type or object properties enables a mapping onto standard accounting software with regards to profit and loss accounts that are related to the business cases that are part of our Retail-H. When accounting related REA events occur (in our prototype these are the events related to the purchase and the sale duality), an accounting record is created and pushed on a "booking stack" (a database table) from where it can be grabbed by standard accounting software. The accounting records are created in a way that typically there is an accounting record per each stockflow instead of per each event—the accounting records are therefore very fine-grained. Manual checks of the booking stack reveiled that the accounting records are created as expected.

VII. Conclusion

A prototypical retail information system was implemented with a generic REA engine at its core that is used to record all occurring business data and that is capable of being configured at runtime. That way the REA engine could be left completely domain agnostic, and specific business models are defined at runtime by using a tree based modeling tool in the browser. We have presented one such domain specific model: the data model of Retail-H. The developed prototype is capable of driving multiple clients (tenants) in parallel, each with its own REA model. Tenants that use the presented Retail-H framework as a basis for their business model can further refine it by adding new agent or resource types, by adding or manipulating participants or stockflows in events, etc.

As all information of the business is recorded in REA structures, and since REA is at its roots an accounting framework, it is possible to generate balance lists "by the press of a button". All that is required is the quering of the relevant events for a given balance and time frame. In REAlist some of this functionality has been implemented by balance lists that show unsettled claims. One could go even further and potentially create a financial statement for a given time frame (currently this is not possible due to some missing structures and functions, as noted below). REAlist also provides an initial set of key performance indicators (KPIs) that are relevant for the retail domain, including sales share, stock coverage, stock turnover, stock turnover rate and turnover period. The values for these KPIs are calculated on demand by iterating through all relevant business cases and executing the corresponding calculations.

Coming back to the Retail-H model: REAlist has modeled and implemented large parts of the structural entities of Retail-H's "H", accompanied by a web frontend that provides a workflow to insert business cases. The basement and the roof of the Retail-H have not been dealt with in that detail: human resources (HR) are not covered apart from the fact that REA agents can be modeled and equipped with master data and that they participate in diverse events. REAlist has not implemented labor costs or HR planning. Cost accounting, general accounting and asset management have been approached in a superficial way that enables some specific functionality but is far from a complete accounting system. Of the three components of the roof only the controlling part has been partly addressed by providing online retrievable KPIs. Caused by the lack of functional support for HR costs, many parts of the roof and the basement are not realized yet, in the current state of the prototype. Improved support for warehousing and HR are the logical next steps in the development of REAlist and will provide an even better data basis for the recording of accountable artefacts and for planning and controlling.

[5]Portable Document Format

Acknowledgment

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.

References

[1] W. E. McCarthy, "The REA accounting model: A generalized framework for accounting systems in a shared data environment," *The Accounting Review*, vol. 57, no. 3, pp. 554–578, 1982.

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

[3] W. S. A. Schwaiger and M. Abmayer, "Accounting and management information systems: A semantic integration," in *Proceedings of the 15th International Conference on Information Integration and Web-based Applications & Services (iiWAS2013)*. New York, NY, USA: ACM, 2013, pp. 346:346–346:352. [Online]. Available: http://doi.acm.org/10.1145/2539150.2539214

[4] R. Haugen and W. E. McCarthy, "REA, a semantic model for internet supply chain collaboration," in *6th Int'l Workshop on Business Object Component Design and Implementation*, 2000.

[5] J. Becker and R. Schütte, *Handelsinformationssysteme*, 2nd ed. MI Wirtschaftsbuch, 2004.

[6] ——, "Reference model for retail enterprises," in *Reference Modeling for Business Systems Analysis*, P. Fettke and P. Loos, Eds. Idea Group Inc., 2007, pp. 182–205.

[7] G. L. Geerts and W. E. McCarthy, "An accounting object infrastructure for knowledge-based enterprise models," *IEEE Intelligent Systems*, vol. 14, no. 4, pp. 89–94, 1999.

[8] ——, "The ontological foundation of REA enterprise information systems," in *Annual Meeting of the American Accounting Association*, 2000, pp. 127–150.

[9] ——, "An ontological analysis of the economic primitives of the extended-REA enterprise information architecture," *International Journal of Accounting Information Systems*, vol. 3, no. 1, pp. 1–16, 2002.

[10] ——, "Policy-level specifications in REA enterprise information systems," *Journal of Information Systems*, vol. 20, no. 2, pp. 37–63, 2006.

[11] ISO and IEC, *Business Transaction Scenarios—Accounting and Economic Ontology*, International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) Std., Rev. ISO/IEC 15944-4:2007(E), 2007, ISO/IEC 15944-4:2007(E).

[12] D. Mayrhofer, "REA-DSL: Business model driven data engineering," PhD Thesis, Vienna University of Technology, 2012.

[13] M. Al-Jallad, "REA business modeling language: Toward a REA based domain specific visual language," Thesis, KTH Royal Institute of Technology, 2012.

[14] V. Meister, *Grundlagen betrieblicher Anwendungssysteme: Integrative Lösungsansätze für die betriebliche Praxis*, ser. Kontakt & Studium. expert Verlag, 2011, no. 703.

[15] N. Gronau, *Enterprise Resource Planning: Architektur, Funktionen und Management von ERP-Systemen (in German)*, 2nd ed., ser. Lehrbücher Wirtschaftsinformatik. Oldenbourg Verlag, 2010.

[16] J. A. Gulla and T. Brasethvik, "A model-driven ERP environment with search facilities," *Data & Knowledge Engineering*, vol. 42, no. 3, pp. 327–341, 2002.

[17] P. Dugerdil and G. Gaillard, "Model-driven ERP implementation," in *2nd International Workshop on Model-Driven Enterprise Information Systems (MDEIS 2006), at ICEIS 2006*, 2006.

[18] A. Dreiling, M. Rosemann, W. M. P. van der Aalst, W. Sadiq, and S. Khan, "Model-driven process configuration of enterprise systems," in *Wirtschaftsinformatik 2005*. Physica-Verlag, 2005.

[19] R. E. Johnson and B. Woolf, "Type object," in *Pattern Languages of Program Design 3*, R. C. Martin, D. Riehle, and F. Buschmann, Eds. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1997, ch. Type Object, pp. 47–65. [Online]. Available: http://dl.acm.org/citation.cfm?id=273448.273453

[20] B. Wally, A. Mazak, B. Kratzwald, C. Huemer, P. Regatschnig, and D. Mayrhofer, "REAlist—a tool demo," in *9th International Workshop on Value Modeling and Business Ontology (VMBO 2015)*, February 2015.