CAPITALISM IN UNIFIED MODELING LANGUAGE

DESIGNING SERVICE-ORIENTED APPLICATIONS THAT UNDERSTAND YOUR BUSINESS

Pavel Hruby

CSC

phruby@csc.com

CSC Papers

2008

ABSTRACT

Have you ever tried to analyze large amounts of data in your enterprise resource planning system, and found it difficult to trace causes and effects of certain transactions or reported data, or found it too cumbersome, or even impossible to customize the application to exactly fit the changing needs of your organization?

This is because all current financial applications are based on the principles established at the end of the 15th century; which are suitable for a different environment than we live in today, and which are unable to use the full potential of information technologies. Consequently, none of the current enterprise resource planning systems offers full traceability of business data, each only has fragments of generic business semantics embedded, and therefore, applications have only limited interoperability with other enterprise information systems. As a consequence, to a large extent, extracting actionable meaning from business data is left to the end user.

A solution to these problems is offered by the REA (Resources, Events, Agents) modeling framework and ontology for business systems. The REA model captures the cause-and-effect relationships between business transactions, and consequently the applications based on the REA principles provide for full traceability of data that influence values of economic resources. This paper explains basic REA principles, and also discusses how to use service-oriented architecture (SOA) to extend the REA model to provide functionality required in specific business areas, and the scalability and customizability of the software applications.

Mastery of REA architectural principles will allow a firm to strengthen its leading position in the development of business solutions. Its customers will benefit from the availability of inexpensive applications with embedded generic business semantics.

Table of Contents

1 INTRODUCTION

This paper describes the REA (resources, events, agents) model, which specifies the fundamental laws of the business domain. Knowing these laws radically enhances the application designers' potential to develop business solutions without omissions, and ensures consistency of software applications from the business perspective:

- Software applications based on the REA model contain more business knowledge than applications developed merely from user requirements, and can therefore advise and guide the users during development and configuration, without restricting the end-users at runtime.

- The application design based on the REA model is concise and easy to understand both for the users of software applications, for consultants, and for application developers. The REA model describes a ubiquitous language ensuring unambiguous communication and understanding among all participants of the software development process.

- The same modeling principles are used across all application areas in the business domain; the sales, procurement, production, marketing, human resources, finance, and other areas are described by a common set of patterns.

- As REA software applications store the primary data about economic resources, all reports and all accounting artifacts are always consistent, because they are derived from the same data; for example, the data describing the sale event is used in the warehouse management, payroll, distribution, finance and other application areas, without transformations or adjustments.

- The REA model provides for more complete, transparent, and up-to-date reporting for business decision makers than reporting based on the accounting artifacts, which dominates in current business applications.

The REA model was originally proposed as a generalized accounting model for use in a shared data environment. It was first published by William E. McCarthy of Michigan State University (McCarthy 1982). Since then, the original REA model, based on the fundamental categories of economic event, economic agent and economic resource, has been extended by McCarthy and Guido Geerts to an ontology for business systems (Geerts, McCarthy 2000a, 2002), and includes additional categories such as commitments and contracts. The REA model became the foundation for several electronic business interchange standards, such as ebXML[1], and UN/CEFACT[2], and Open-edi (ISO/IEC 15944-4). The REA model has established a solid reputation among business analysts and teachers of enterprise information systems as a conceptual framework for modeling business systems and as a business analysis tool. This reputation is largely based on the ontology's ability to explain why economic transactions occur; that is to reveal the cause-and-effect relationships between economic transactions. This unique feature distinguishes the REA model from alternative business ontologies such as e[3]value

---

[1] Electronic business eXtensible markup language, http://ebxml.org/

[2] United Nations Centre for Trade Facilitation and Electronic Business, http://www.unece.org/cefact/

(Gordijn 2002) and eBMO (Osterwalder 2004), as well from alternative business process modeling methods such as BPML.

This paper is structured as follows. In the first part, state-of-the-art, outlines common problems that we experience today when developing and using business software applications, and how these problems can be solved using the REA model. The second part, What Is the REA model, describes REA concepts, domain rules assuring that the application model is sound from the business perspective and illustrates, using an example called Joe's Pizzeria, how to apply the REA model to form the backbone of the application model

Knowing the REA model is useful but not sufficient to build a business application; similarly, knowing only Maxwell's laws is not sufficient to build a radio transmitter and receiver. Therefore, the REA model can (and should) be extended using a number of patterns that comprise functionality necessary to build business applications that meet specific business needs. We describe the patterns that extend REA in the section Extending the REA model with Services. The last part, Why We Do not Have REA Software, Yet, gives references and tips to a reader who would like to know more.

2 THE STATE OF THE ART

This section outlines several common problems that we experience today with enterprise resource planning systems, and how the REA model can contribute to solving these problems.

2.1 THE END OF USERS

The expression "the end of users" has been used by Mary Beth Rosson's keynote at OOPSLA 2005[3]. She observed that many end users would like to customize their business applications themselves, without needing a specialist (and therefore become occasional "developers"). The success of Microsoft Excel as a business application can be partially explained by its empowering of users to model business processes themselves.

The following is a traditional scenario for implementing a new enterprise business solution in a corporation:

1. A consultant works with users to describe the corporate business processes to be supported by a software solution.

2. A team of developers receives the consultant's description, but the developers have trouble understanding the business terminology and find the description too informal to use for implementing the system.

3. The developers write their own system specification from a technical point of view.

4. When the system specification is presented to the users, they do not quite understand it because it is too technical. They are, however, forced to accept it as a contract for system development.

---

[3] http://www.oopsla.org/2005/ShowEvent.do?id=407
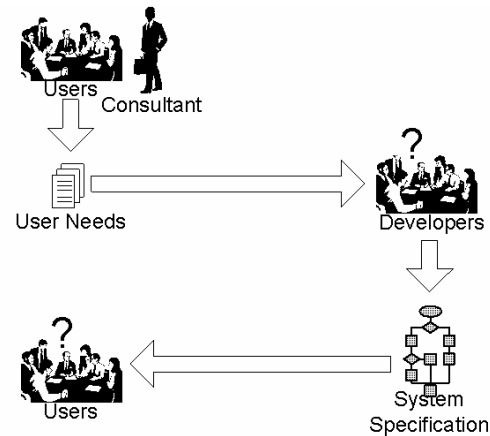
Fig. 1. Users' needs are not suitable as a system specification. On the other hand, developer's system specification is not fully understood by users.

This approach can easily result in a system that does not meet user requirements because often the users, the consultants and the developers don't speak the same language. Such communication problems can make it difficult to turn a description of business processes into a technical software specification that all parties can understand. In addition, because a technical system specification that is not fully understood by the actual users of the system is used, business applications become difficult to use.

When the solution is delivered and runs, users lack information about actual efficiency of the implemented business processes because the indicators or statistics are closely related to the software solution, and their business interpretation might be confusing. Improving business processes is difficult because they are defined by means of a system specification that is not understood by actual users of the system.

Benefits of Using the REA Model

As a means of solving this problem of understanding, REA concepts can be used as a ubiquitous language[4] that consultants, developers and end users alike use to define, analyze and improve the business processes. The team will design and specify the system in a single creative procedure that will approach things from the user's point of view and use language that is identical to the language of the user's daily work.

---

[4] The need for ubiquitous language in software design has been described in (Evans 2005). This book also showed that a design becomes much better if it is expressed in users' terms, not in technical terms.
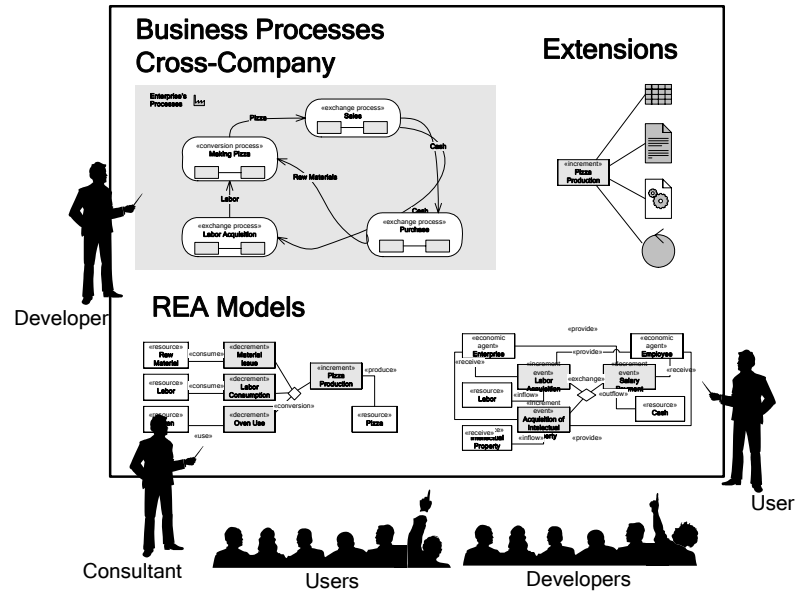
Fig. 2. REA models are intuitive for the user and are based on his perspective and terminology. In addition, they have precise semantics so the models are executable.

The REA model describes the company's business processes in a natural way for the users and therefore increases efficiency and productivity of the parties involved in system design, including the creation and customization of business processes by users and software non-specialists. The model offers an advanced authoring environment, where content is tagged with semantic metadata. For example, the environment will map entities of the customer's world like Shipment and Payment to the REA ontological category Economic Event.

The value chain model, specified in the REA modeling framework, can be used as a catalogue of common business processes. The catalogue reduces and perhaps even eliminates the work of implementing a system that has been specified in the catalogue. Users themselves, without intervention of a consultant or developer, are able to customize their business processes using the interactive modeling environment, and support distributed business processes. The applications will be able to integrate with company-wide or cross-company business processes by means of the REA business ontology.

2.2 DOUBLE-ENTRY BOOKKEEPING DOES NOT MEET DEMANDS FOR MODERN INFORMATION ARCHITECTURES

Double-entry bookkeeping is the standard system used by business organizations to record financial transactions. The results of an organization's business operations are represented by accounts, each of which reflects a particular aspect of the business as a monetary value. This method is called double-entry because each transaction is entered twice. Each debit value must have a corresponding credit value, and all transactions must "balance" so that when you add up all the debit balances, the total must be the same as the total of all the credit balances. Double-entry bookkeeping is a method originating from the end of the 15th century, when the number of transactions was relatively low, the transactions were

processed using tools such as pen and paper, and negative numbers had not yet been discovered.

Despite its historical success, this method has its limits. The value of transactions, and their balance, is the main method of establishing traceability between them. This is only indicative and provides a partial level of traceability. This was a feasible way when using tools such as pen and paper, but this method does not exploit all possibilities offered by modern data processing technologies.

Along with increased enterprise sizes, enterprise information systems are becoming more complex, and it is more and more difficult to process the relevant data for enterprise management using the 15th century method. Unfortunately, it also opens up an increasing number of ways to manipulate data. The Enron case in the USA showed the scale such data manipulation can reach.

We see a demand for an enterprise resource planning system that would be designed from scratch for use with information technologies. This system would enable full traceability of **all**, not just some, activities that influence the value of the enterprise's resources. The value of the enterprise's resources could be calculated **on demand** and be available whenever required, not only once a year, a month or two after closing the financial books.

Benefits of Using the REA Model

It is clear that computer systems are able to provide a much greater level of traceability between transactions than only by comparing whether their monetary values are equal. Relational databases, for example, can represent transactions as records and trace transactions by means of the relationships between them.

The REA ontology establishes the theoretical foundation for full traceability in enterprise resource planning systems by defining the fundamental relationships that may exist between transactions. Moreover, it also defines fundamental entities, some of which are not present in double-entry bookkeeping, such as agreements and contracts. People who are not experts in accounting are often surprised by the fact that the monetary value of a company's orders and other contracts, a very important indicator for business decision makers, is not represented in the chart of accounts, the central document of double-entry bookkeeping.

An REA-based application may create the same set of accounts as the double-entry system and provide the same required functionality, so the legal requirements for accounting software can easily be met[5]. In addition, an REA-based application offers a consistent and complete data model from which business reports (not only financial reports) can be created. As the REA application has embedded business semantics, the application can provide relevant data needed for every decision maker in any given situation, and will consequently support context-aware or personalized distribution and presentation of business data.

---

[5] Thanks to full traceability of all transactions that influence the value of economic resources, REA software naturally support demands set by the Sarbanes-Oxley Act. In addition, full traceability is achieved for all transactions, not only financial transactions.

2.3 MODEL-DRIVEN DESIGN

Model-Driven design is an approach to developing software applications by modeling the solution instead of writing code.

The state-of-the art tools for model-driven design can deliver that goal only partially. Most tools cannot produce fully functional executable code (unless the model is overly complex); they often generate a code skeleton, which might include executable actions but is to be supplemented by hand-written code. Debugging almost always occurs at the code level and not at the model level.

One of the reasons for this is that generic modeling languages such as UML, BPMN, IDEF0, BPEL and YAWL do not contain enough domain-specific semantics. Generic modeling languages have many advantages (such as a very large scope), but a key drawback from the perspective of model-driven design is that they are not specific enough to be compiled into executable software without additional hand-written code, unless the model is overly complex and hard to understand by non-technical users.

Current trends show that the objective of automatically creating a software application *only* by modeling can be achieved if the modeling language is not generic but domain-specific. As the domain-specific language has embedded domain semantics, this makes the model much simpler, and a tool can be used to validate the model. Examples of successful domain-specific modeling tools are MATLAB and SIMULINK, in the domain of linear dynamic systems.

Alternative business process ontologies, currently being developed at many research institutions (such as e^3value and eBMO ontologies), attempt to address the problem of specificity. These ontologies can successfully model **what**, **when**, **where** and sometimes **how**, but they do not capture the cause-and-effect relationships between business transactions.

> The REA model is the only known business modeling method that can provide the answers to **why** business processes occur, reveal the cause-and-effect relationships between economic transactions, and provide for full traceability of all transactions that influence the values of economic resources.

An example is: "Why do I need to pay this shop?" The answer, which an REA-based application can provide, is: "Because you received the goods." This answer cannot be given by any alternative business modeling method or ontology. None of the approaches, such as Use Case model, Data model, IDEF0, BPMN, BPEL, YAWL, e^3value, eBMO, or any other state-of-the-art business modeling approaches, can answer the question of why an enterprise performs its business processes, and why a specific modeling element is part of the model, because the business interpretation of the modeling elements, and business validation of the model, are not available to a tool and are left solely to the modeler.

Benefits of Using the REA Model

Compared to generic modeling languages that use general-purpose concepts such as activities, tasks, data entities and objects to express business processes, the REA model uses five specific concepts to create a model:

- economic resource

- economic agent

- economic event

- commitment

- contract

These concepts, together with their relationships, are described in detail in the section "What Is the REA Model?"

REA models are fully executable. The fact that REA modeling elements are more specific than elements such as activity or data entity radically (and surprisingly) increases the amount of information in the model, while preserving its simplicity. The amount of information in the model is so high that the model can be compiled into executable code – that is, an enterprise resource planning application – without needing any additional information or modifications from a software developer (this has been proved by a project run by Navision from 1999-2003).

The REA ontology contains rules for formulating well-formed models of enterprise processes that a tool can use to make a syntactic check of the model based on business semantic rules. For example, all well-formed REA models obey a fundamental rule that every increase of a resource value for an economic agent is always paired with some decrease of the value of some of its resources; that is, there is no increase in the resource value for free. The relationship between increment and decrement event represents the causality between economic events (business transactions), and enables a tool to deduce **why** economic events occur or should occur.

Another example of a rule for validating consistency of the model from an economic perspective is that for every economic resource there must exist at least one economic event explaining how the economic agent disposed of or consumed this resource. Likewise, for every economic event that increases the value of an agent's resources, there must exist an economic event that decreases the value of the agent's resources, and for every economic event there must be two economic agents: the recipient and the provider of the economic resource specified by the event.

2.4 EVOLUTION OF LARGE SYSTEMS

Enterprise resource planning systems are evolving over time because the business processes they support are changing. This applies to product lines as well as individual installations. One of the challenges affecting systems that must evolve over time is to determine the stable core, comprising the fundamental principles of the user's business and the parts of the system that might change, for example, together with changing technologies.

None of the state-of-the art enterprise resource planning systems makes that distinction sufficiently clearly. The consequence is that as the system evolves, it becomes increasingly difficult to manage the dependencies between different features and added functionality. For example, it is known that in the case of

Microsoft Dynamics NAV, one of the successful enterprise resource planning systems for the mid-range market, customers can decide between a customized solution but not being able to upgrade to future versions, or being able to upgrade but having the application only with generic logic. A heavily customized Microsoft Dynamics NAV solution is very expensive to upgrade.

There are several approaches attempting to structure the business systems by describing the fundamental modeling entities, such as **archetypes** (Coad, Lefebvre, DeLuca. 1999) and **pleomorphs** (Arlow, Neustadt 2003), for the business domain, and many **business patterns** on more detailed levels; my favorite books include (Fowler 1996), (Hay 1996), (Silverstone 1997), (Marshall 2000) and (Evans 2003). The patterns and modeling entities described in these books can be expressed in terms of the REA concepts. These patterns are more specific, as they focus on certain subdomains within the business domain. They are excellent generalizations of many existing software solutions, but they all miss the fundamental skeleton (such as that their primary purpose is to support planning, monitoring and controlling exchanges of economic resources between trading partners).

Benefits of Using the REA Model

The REA model specifies the criteria distinguishing the stable and the evolvable parts of the system expressed in users' business terms. REA concepts determine the stable core of the system, defining the very fundamental principles of an enterprise's business and its purpose, which is often independent of the supporting technologies. This stable core can be extended by services that are designed to be inexpensive to change in time to reflect changing company business needs. This is schematically illustrated in Fig. 3.
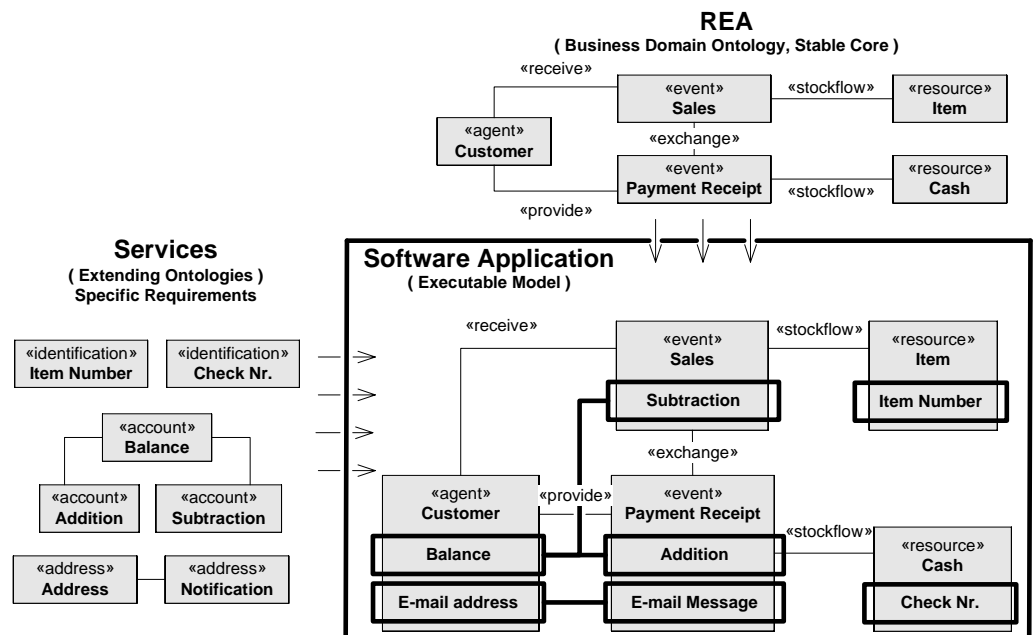

Fig. 3. The REA ontology can be extended by other ontologies to reflect specific and changing requirements.

Such an approach has been practically proved by a project at Navision from 1999-2003 that had very positive results; the product has been shipped in a beta version (the project was stopped in 2003 after the acquisition of Navision by Microsoft.)

3 WHAT IS THE REA MODEL?

The REA model is an ontology and modeling framework that specifies classification of the concepts in a business domain, derived from the fundamental idea that business is based on exchanges of economic resources between trading partners. In this paper I describe REA concepts informally, and illustrate them on Joe's Pizzeria example.

3.1 CONCEPTS

There are several concepts that are present in almost all business software applications. Understanding these concepts makes it much easier to design business applications, to ensure that they do not violate the domain rules, and to adapt the applications to changing requirements without the need to change the overall architecture.

These concepts are known as the REA model (Resources, Events, Agents). Fig. 4 illustrates the most fundamental REA concepts, which are economic resource, economic agent, economic event, commitment, and contract.
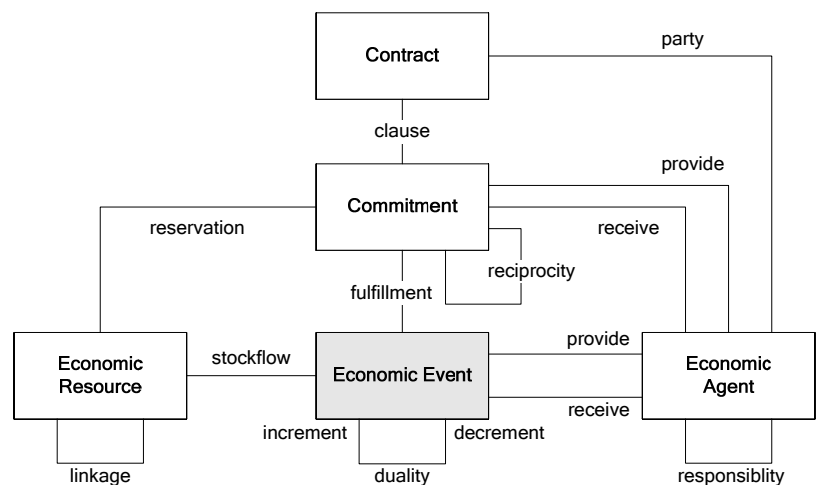

Fig. 4. Fundamental REA concepts

Economic Resource is a thing that is scarce, and has utility for economic agents, and is something users of business applications want to plan, monitor, and control. Examples of economic resources are products and services, money, raw materials, labor, tools, and services the enterprise uses.

Economic Agent is an individual or organization capable of having control over economic resources, and transferring or receiving the control to or from other individuals or organizations. Examples of economic agents are customers, vendors, employees, and enterprises. The *enterprise* is an economic agent from whose perspective we create the REA model.

Economic Event represents either an increment or a decrement in the value of economic resources that are under the control of the enterprise. Some economic events occur instantaneously, such as sales of goods; some occur over time, such as rentals, labor acquisition, and provision and use of services.

Commitment is a promise or obligation of economic agents to perform an economic event in the future. For example, line items on a sales order represent commitments to sell goods.

Contract is a collection of increment and decrement commitments and terms. Under the conditions specified by the terms, a contract can create additional commitments. Thus, the contract can specify what should happen if the commitments are not fulfilled. For example, a sales order is a contract containing commitments to sell goods and to receive payments. The terms of the sales order contract can specify penalties (additional commitments) if the goods or payments have not been received as promised.

The fundamental idea of the REA model is

> If an enterprise wants to increase the total value of resources under its control, it usually has to decrease the value of some of its resources.

An enterprise can increase or decrease the value of its resources either by **exchanges** or by **conversions**.

- **Exchange** is a process in which an enterprise receives economic resources from other economic agents, and it gives resources to other economic agents in return.

- **Conversion** is a process in which an enterprise uses or consumes resources in order to produce new or modify existing resources.

The data associated with exchanges and conversions are the primary business data about the enterprise in REA software applications. Accounting artifacts such as debit, credit, journals, ledgers, receivables, and account balances are derived from the data describing the exchanges and conversions. For example, the quantity on hand for an inventory item can be calculated from the difference between the purchase and sale events, or between the production and consumption events, for that inventory item.

For comparison, in most current business software applications, whose paradigms are derived from double entry accounting, it is the opposite – they focus on the accounting artifacts, and economic data is derived from them. This, in some sense, puts the consequences before the cause and makes the models more complicated.

The fact that the REA applications operate on primary and raw economic data explains why they offers a wider, more precise, and more up-to-date range of reports than double-entry accounting applications that operate on derived accounting data.

3.2 JOE'S PIZZERIA

We will create an REA model for Joe's Pizzeria

Joe makes revenue by selling pizza to his customers. Joe's Pizzeria has employees whose task is to sell pizzas, as well as to produce pizzas from raw materials such as dough, tomatoes, cheese, pepperoni and other toppings. There are also other things necessary to produce pizza, such as the oven where the pizza is baked, electricity consumed to heat the oven, various kitchen equipment and many other things. Joe is interested in tracking information about some of them; in the REA model the things that Joe is interested in planning, monitoring and controlling are called economic resources. Joe has decided that the economic resources that will be included in the business software application are the Pizza, Cash, Labor of the employees, and Raw Materials and Ingredients for producing pizza.
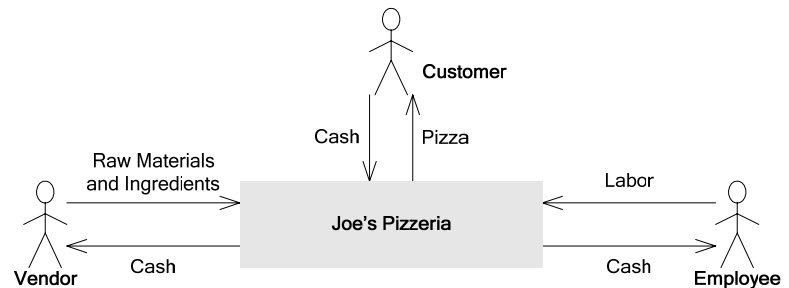
Fig. 5. Trading partners of Joe's Pizzeria

Trading partners of Joe's Pizzeria are customers, vendors and employees. They are capable of controlling economic resources; therefore, in the REA application model the Customer, Vendor, Employee, and Joe's Pizzeria are economic agents; see Fig. 6.

The main business processes of Joe's Pizzeria (see Fig. 6) are selling pizza to the customers (the Sales process), purchasing raw materials from the vendors (the Purchase process), and purchasing labor from the employees (the Labor Acquisition process). We will construct the REA model for each process.
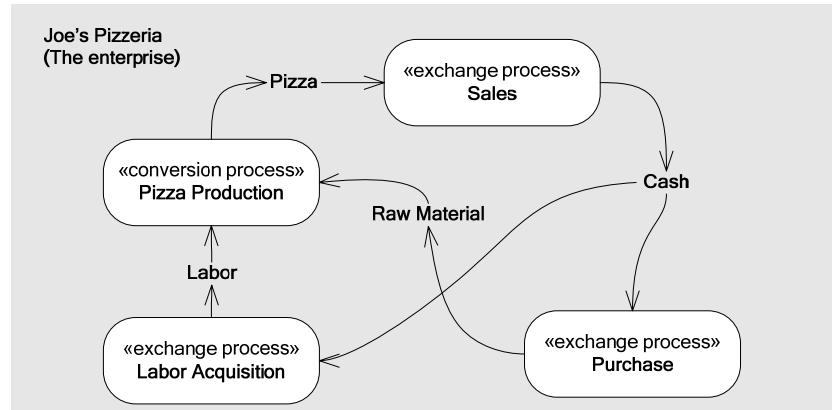
Fig. 6. Business processes of Joe's Pizzeria

3.2.1 SALES PROCESS

The process of selling pizza to the customers is essentially an exchange of pizza for cash; Joe's Pizzeria gives *Pizza* to the customer, and receives *Cash* in return. For Joe's Pizzeria, the *Sales* process represents an outflow of *Pizza* and an inflow of *Cash*; see Fig. 7.
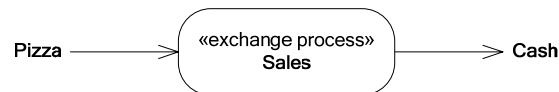

Fig. 7. Selling pizza is an exchange of pizza for cash

The REA model for the process of selling pizza is illustrated in Fig. 8. *Joe's Pizzeria* and the *Customer* are economic agents, and the *Pizza* and *Cash* are economic resources. One economic event is the transfer of ownership of the *Pizza* from *Joe's Pizzeria* to the *Customer* (we call this event *Sale*); in this transaction *Joe's Pizzeria* provides *Pizza*, and *Customer* receives it. Another economic event is the transfer of ownership of *Cash* from the *Customer* to *Joe's Pizzeria* (we call it *Cash Receipt*); in this transaction the *Customer* provides *Cash*, and *Joe's Pizzeria* receives it.

For *Joe's Pizzeria*, the *Sale* event (the transfer of ownership of the *Pizza* to the *Customer*) is a decrement event, because it decreases the value of the resources under the control of *Joe's Pizzeria*. The *Cash Receipt* is an increment event, because it increases the value of the resources under the control of *Joe's Pizzeria*. The terms decrement and increment are relative to the model viewpoint; they depend on the economic agent which is in the focus of the model. If we modeled the same process from the perspective of the *Customer*, the transfer of pizza would be an increment (would be called Purchase) and the transfer of cash would be a decrement event (would be called Payment or Cash Disbursement).

The REA model of the sales process in Fig. 8 focuses on the core economic phenomena, and therefore it covers many special cases. For example, most customers pay when they purchase pizza, but some customers may receive an invoice, and pay for all their purchases in a certain period at once. If the case of

Internet sales, customers must provide their credit card information before the pizza is delivered, and Joe's Pizzeria receives money from the customer's bank later. When the sale occurs in the restaurant, the customers pay after they get pizza, either using cash or a credit card.
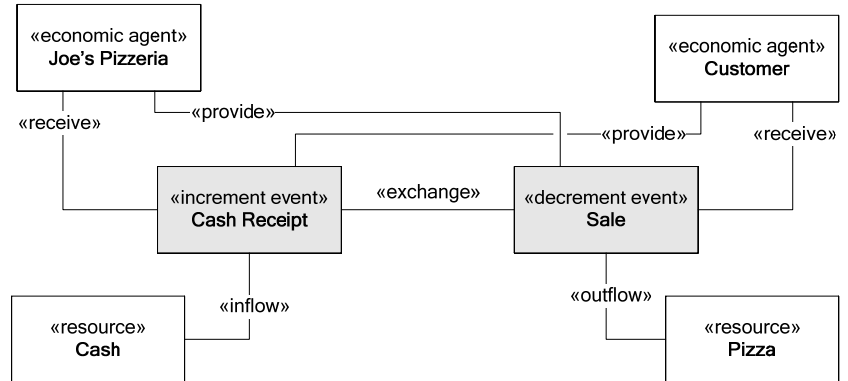

Fig. 8. The REA model for Joe's Pizzeria sales process

All these cases are covered by the model in Fig. 8; this is very useful if we would like to create a robust skeleton of a software application.

Customers may order pizza over the Internet. In this case, a software business application creates an electronic *Sales Order*, which specifies a commitment of *Joe's Pizzeria* to sell a specified *Pizza* to the *Customer*, and a commitment of a *Customer* to pay for the *Pizza* a specified amount of *Cash*.

The *Sales Order, see* Fig. 9*,* is an example of a contract between the economic agents *Joe's Pizzeria* and the *Customer*. The *Sales Line* and the *Payment Line* are not economic events; they are commitments to perform the economic events in well-defined future. The *Sales Line* is a commitment to perform the event *Sale*, and the *Payment Line* is a commitment to perform the event *Cash Receipt* in the future.
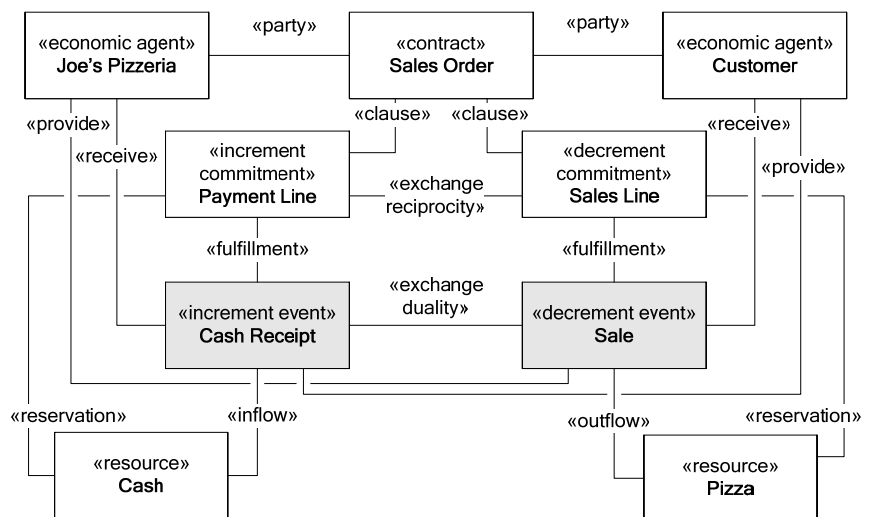

Fig. 9. The REA model for the sales process with sales order

The *Sales Order* often contains terms specifying what should happen if the commitments are not fulfilled, such as when the payment arrives late, or the customer is not satisfied with the pizza. The fact that a contract can be represented as a computer model is important for automatic tracking of the state of the contract at runtime, and also for computer-assisted evaluation of complicated financial contracts.

3.2.2 PURCHASE PROCESS

When Joe's Pizzeria purchases tomatoes, cheese, pepperoni, flour and other raw materials, it essentially exchanges the raw material for cash. *Vendor* gives *Raw Material* to *Joe's Pizzeria*, which gives it *Cash* in return. For Joe's Pizzeria, the *Purchase* process represents an outflow of *Cash* and an inflow of *Raw Material*, see Fig. 10.
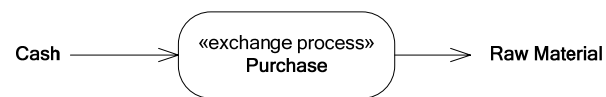

Fig. 10. Purchasing raw material is an exchange of raw material for cash

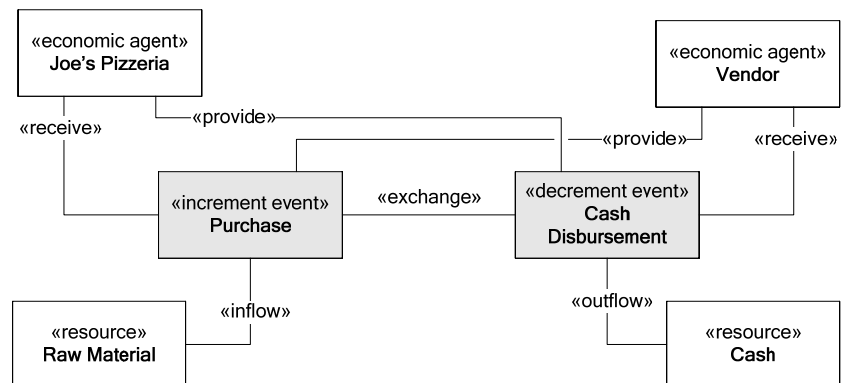The REA model for the process of purchasing raw material is illustrated in Fig. 11.

Fig. 11. The REA model for the purchase process

The *Vendor* and *Joe's Pizzeria* are economic agents, the *Raw Material* and *Cash* are economic resources. The transfer of ownership of the *Raw Material* from the *Vendor* to *Joe's Pizzeria* is an increment economic event (we call it *Purchase*), and the transfer of ownership of *Cash* from *Joe's Pizzeria* to the *Vendor* (we call it *Cash Disbursement*) is a decrement economic event; the increment and decrement are from *Joe's Pizzeria* perspective.

Similarly as for the REA model for sales, the REA model for purchases covers many special cases. Some raw materials can be paid by check or bank transfer; some can be made in different currencies. There can be several purchases paid using a single payment, and a single purchase can be paid in several installments.

The model tracks the information about which purchases correspond to which cash disbursements, but abstracts from technical details and does not specify the order of these transactions. Again, this is useful if the skeleton of a software application is based on this model, because it does not have to be changed if some technical aspects of the purchase process change.

3.2.3 LABOR ACQUISITION PROCESS

Joe's Pizzeria employees provide their work (they produce and sell pizzas during specified periods of time) and receive their salary in return. Labor acquisition is essentially an exchange of *Labor* (the worked hours) for *Cash*. *Employee* sells his labor to *Joe's Pizzeria*, which gives him *Cash* in return. For Joe's Pizzeria, the *Labor Acquisition* process represents an outflow of *Cash* and an inflow of *Labor*, see Fig. 12.
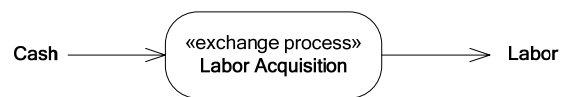

Fig. 12. Labor acquisition is an exchange of worked hours for cash
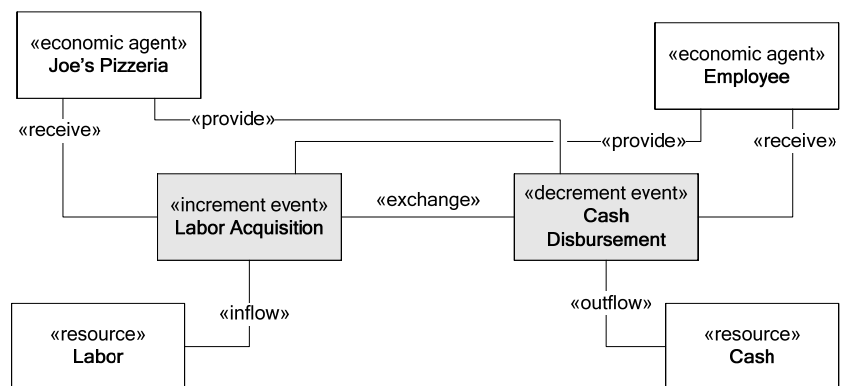

Fig. 13. The REA model for the labor acquisition process

The REA model for the labor acquisition process is illustrated in Fig. 13. The *Employee* and *Joe's Pizzeria* are economic agents; the *Employee* provides *Labor* and receives *Cash*, and *Joe's Pizzeria* provides *Cash* and receives *Labor*. *Labor* (the worked hours) and *Cash* are economic resources. The *Labor Acquisition* is an economic event that occurs over periods of time (during the employee's working hours), while *Cash Disbursement* is an instantaneous event that occurs once a week or month when the *Employee* receives his paycheck.

The REA model in Fig. 13 can be applied to many forms of acquiring labor; it can be applied for full employment, temporary work, consulting, as well as for work acquired according to various other forms of contracts.

3.2.4 SUMMARY

The REA model focuses on the core economic phenomena and abstracts from technical and implementation details. This has several advantages.

Firstly, the REA model abstracts from the technical aspects of the transfer of the resources. Cash can be transferred as bills and coins, as a check or as a credit card transaction. Customers can pick pizza themselves, or pizza can be delivered to their address. For all these cases we can apply the same REA model, which does not have to be modified even if the technical infrastructure supporting the business changes.

Secondly, the REA model abstracts from the order in which the economic events occur. Usually, pizza is paid at about the same time as it is given to the customer, but sometimes it is paid for beforehand, and sometimes it can be paid by credit card and there is a significant delay between the sale of pizza and the transfer of cash. If the business process was specified as a scenario consisting of a sequence of events, the business application would support only the scenarios identified at design time. The REA model allows the business application to flexibly record everything that actually happened. The actual order of events emerges at runtime, rather than being specified at design time.

Thirdly, for each REA model apply certain rules: each increment must be related to a decrement, each economic event must have a provider and recipient agent, and each resource must be related to both increment and decrement. Therefore, application designers can ask relevant questions leading to the discovery of missing information in the user requirements, and can construct the model even if the initial specification is incomplete.

The three illustrated models for the business processes *Sales*, *Purchase* and *Labor Acquisition* have many common features. They all model the transactions between Joe's Pizzeria and its trading partners as exchanges of economic resources.

These models can be generalized into a model at a higher level of abstraction, illustrated in the next chapter. The models for sale of pizza, purchase of raw materials and labor acquisition are examples of the *REA EXCHANGE PROCESS PATTERN*.

3.3 REA EXCHANGE PROCESS PATTERN

Trade is the voluntary exchange of goods, services, or money

Context
You are an application designer developing a business application. You are trying to create an object model of a business application and struggling to find the right structure for the model and the right relationships between entities in the model. You know the user requirements; they can be in a written document or non-written

information obtained by an ongoing dialog with the users; but you know the requirements are incomplete. You want to know the right questions to ask to better understand the application domain. You also want the model to be consistent and robust enough for future changes in user requirements.

Problem

How does one create a robust skeleton of an object-oriented model for interactions between the enterprise and it trading partners? User requirements are not a sufficient source of information, because they are known to be incomplete, often contradictory, and to change over time, and it is often impossible to find what requirements are missing. Shortly, you would like to create a business application that will satisfy even some of user requirements that have not been communicated to you.

Forces[6]

The REA exchange process pattern resolves the following forces:

- The modeled software application should provide information about how the interactions between the enterprise and its trading partners change the value of the economic resources of the enterprise. The application should keep track of the increases and decreases of the value of the resources that are under the control of the enterprise, and should record which resources were exchanged for which others.

- Application designers want to concentrate on the fundamentals of the users' business, and separate those requirements which are likely to change. The fundamentals are often so obvious to the users of business applications that they do not communicate them, and they remain hidden until late stages of software development.

- The model should be consistent with the business domain rules. Application designers want to ensure that the model is consistent, complete, and correct, with respect to the domain rules.

- Application designers want to include business semantics into the entities in the application model. Semantics based only on the names of the entities is not good enough because it relies on common knowledge, and common knowledge is not available to software applications.

Solution

Model the interactions between the enterprise and its trading partners as *exchanges* of economic resources.

Each exchange consists of at least one *increment economic event* that increases the value of a resource of the enterprise by transferring rights to the resource to or from other economic agents. Every increment event is related to at least one *decrement economic event* that decreases the value of a resource of the enterprise by transferring rights to the resource to or from other economic agents. We call the

---

[6] In the pattern literature the term **forces** is used for the constraints that restrict the solution of the problem, requirements, and properties that the solution should have.

relationship between the increment and decrement economic events *exchange duality*, or in short, *exchange.* The *exchange duality* is a many-to-many relationship, indicating that in the application model there must be at least one decrement for each increment, and vice versa. Therefore, the exchange duality in the application model can be an n-ary relationship, that relates several increment and decrement entities.

In order for an exchange process to add value, the overall increase in value of the resources related to the increment events should be greater than the overall decrease in value of the resources related to the decrement events.
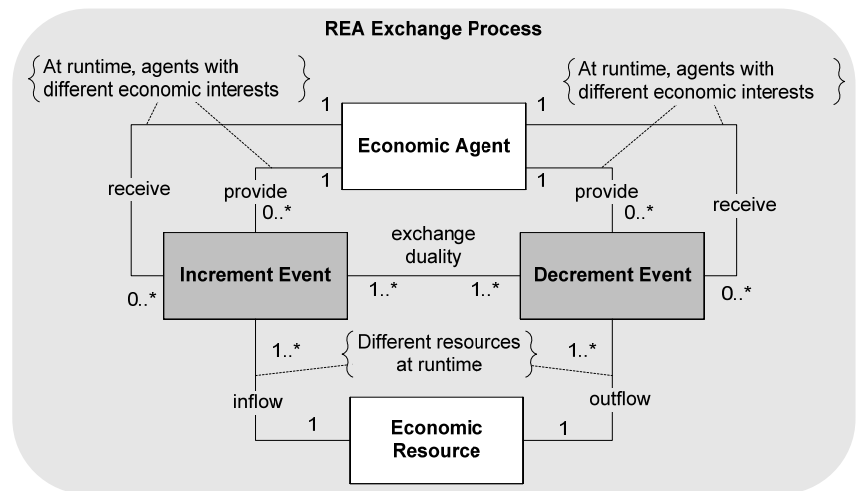

Fig. 14. The REA exchange process

Each *economic event* is related to an *economic resource*, see Fig. 14. The relationship between an increment and a resource is called *inflow*, the relationship between a decrement and a resource *outflow*. In the application model there must be exactly one *economic resource* for each *economic event*, and at least one *increment* and one *decrement* for each *economic resource*.

Each economic event is related to two *economic agents*. The *economic event* in the exchange process transfers rights to the *economic resource* from one agent to another. When the event occurs, the *provider agent* loses rights to the resource, and the *recipient agent* receives the rights. In the application model for each economic event there must be at least one *provider* and at least one *recipient agent*. For each *agent*, there can be zero or more *economic events*.

Note that the model in Fig. 14 determines the rules for constructing the application model. The application model determines the structure of the runtime instances.

The following domain rules apply for any application model describing the REA exchange process.

> Every increment economic event must be related by exchange duality to a decrement economic event, and vice versa.
>
> Every increment economic event must be related by inflow relationship to an economic resource.
>
> Every decrement economic event must be related by outflow relationship to an economic resource.
>
> Every economic event must be related by a provide relationship to an economic agent, and by a receive relationships an economic agent. At runtime, these two agents must represent entities with different economic interests.

Resulting Context

The domain rules in this pattern allow application designers to derive new facts from the facts provided by the users, and to formulate questions leading to the discovery of new facts. Therefore, a business application can meet most or all user needs, even if the user requirements and the designers' knowledge of the user needs are incomplete.

Note that at runtime, for some period of time, there might exist an instance of an increment event that is not paired in exchange relationship with a decrement event. This temporary imbalance results in a claim between economic agents. The claim can be materialized, for example as an invoice. The concept of a claim is described in the Hruby et al. (2006).

3.4 HOW JOE'S PIZZERIA OBTAINS PIZZA

The REA EXCHANGE PROCESS pattern does not apply, because Joe's Pizzeria does not obtain pizzas from its trading partners

3.4.1 PRODUCING PIZZA

Joe's Pizzeria produces pizza from *Raw Materials* such as dough, pepperoni, tomatoes and cheese, by using an *Oven* and by consuming *Labor*. The process of producing pizza is essentially a conversion (transformation) of the *Raw Material*, *Labor* (the worked hours) and the *Oven* (the time when the oven has been used) into a *Pizza,* see Fig. 15. The *Raw Materials* become part of *Pizza*, they are

consumed during production. Employee's *Labor* is also consumed; the time when the employee has worked on pizzas is gone when the pizza is finished, and is not available anymore. On the other hand, the *Oven* can be used again, although it might need some cleaning and maintenance after a *Pizza* has been baked.

In principle, there are also other resources required to produce pizza, such as the kitchen in the building in which Joe's Pizzeria is located, heating of the building, and maintenance of the oven. Joe has decided he is not interested in tracking how they are transformed into each *Pizza*. Therefore we do not model them as economic resources in this process.

However, Joe might decide to include the resources such as maintenance of kitchen equipment and heating of building in the model, and consequently, REA software would be able to determine how each of them contributes to the value of each individual pizza. This is not possible in any solutions based on sustaining technologies, where use and consumption of such resources are considered as "overhead".

Fig. 15. The pizza production process

The REA model for pizza production is illustrated in Fig. 16. The *Material Issue*, *Labor Consumption*, and *Oven Use* are decrements of resources, because they decrease the value of the *Raw Material*, *Labor* and *Oven*. The *Pizza Production* is an increment event, because it creates a new resource with a positive value.
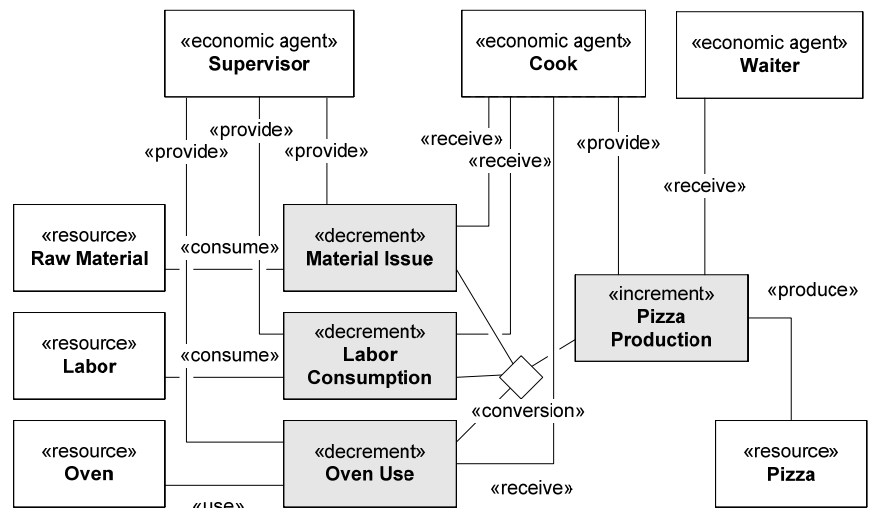

Fig. 16. The REA model for the pizza production

The economic resources *Raw Material*, *Employee Labor* and *Oven* are under the control of the employees *Supervisor*, *Cook* and *Waiter*. The employees physically control the resources on behalf of Joe's Pizzeria, but they do not own them, neither

do they have any legal rights to these resources; the model in Fig. 16 illustrates that the economic agent *Supervisor* issues the *Raw Material* to the agent *Cook*, who bakes a *Pizza* and passes it to the *Waiter*. The *Supervisor* also provides *Oven* to the *Cook* to bake a *Pizza*. To explain who controls *Labor* requires deeper analysis (see the section on Labor in the Modeling Handbook): the *Supervisor* controls *Cook's Labor*, he assigns a task to the *Cook*; the *Cook* consequently takes of the control of his *Labor* and consumes it to produce a *Pizza*.

3.4.2 SUMMARY

The REA model focuses on the core economic phenomena and abstracts from technical aspects of the conversion. This has several advantages.

The model answers the question as to which economic resources have been used, consumed and produced during the process. The economic events provide the information on when, where and how the changes of the resources occurred, and the economic agents provide the information on who controlled the economic resources during these changes. This is the information the business decision makers need in order to plan, monitor and control the economic resources.

The REA model does not imply any restrictions on the time order in which the economic events occur. If the users of a business application wish to specify the desired order of events, the model can be extended using commitments (described in the *SCHEDULE PATTERN*) to specify when the events should occur. However, the model can still record what actually happened, and thus determine the difference between the schedule and the actual production.

In addition to producing pizza, Joe's Pizzeria performs additional activities in order to keep the company running. Cleaning of the restaurant and maintenance of the equipment are the examples. If Joe schedules the pizza production in order to purchase the right amount of raw materials, or if he has an accountant who keeps his financial books, the planner's and accountant's labor are transformed into the services that, as their end result, make Joe's Pizzeria a better company. The cleaning, maintenance, planning, accounting are essentially conversions of labor and tools into other economic resources.

The pizza production, and the abovementioned processes are examples of a pattern, the *REA CONVERSION PROCESS.*
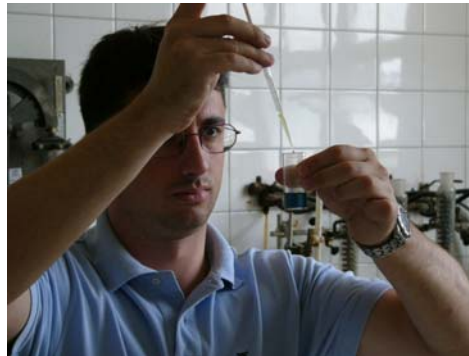
3.5 REA CONVERSION PROCESS PATTERN

Photo by Ulrik de Wachter

Conversion is a physical, structural, or design change or transformation from one state or condition to another

Context

You are an application designer developing a business application. Among the business processes of the enterprise, there usually are one or more processes that create new products or services, or add value to the existing ones. These processes might be specified by the users of a business application, but you know the user requirements are incomplete. You want to know the right questions to ask to better understand the application domain. You also want the model to be consistent, and robust against future changes in user requirements.

Problem

How does one create a robust skeleton of an object-oriented model for a business process that creates new products or services, or adds value to the existing ones? User requirements are not a sufficient source of information, because they are known to be incomplete, often contradictory, and to change over time, and it is often impossible to find out what requirements are missing. In short, you would like to create a business application that will satisfy even some of the user requirements that have not been communicated to you.

Forces

The solution to this problem is influenced by four forces.

- The model should provide information about how the process of creating and modifying resources influences their value, and when the value has been changed.

- The model should provide information about who was responsible for the resources and when.

- The model should capture the fundamentals of the users' business, and filter out those user requirements that are likely to change.

- The model should be consistent, complete, and correct with respect to the business domain rules.

Solution

Model the process that creates new products or services or adds value to the existing ones as a *conversion* of some economic resources to others. During the conversion, the enterprise uses or consumes economic resources in order to produce the resources of the same or another kind.

Each conversion consists of at least one *increment economic event* that increases the value of the resource by modifying its features, and at least one *decrement economic event* that decreases the value of a resource by modifying its features. The *increments* and *decrements* in the conversion processes typically occur over a period of time.

Each *increment event* is related to exactly one *economic resource* by a relationship called *produce*. The *produce* relationship means that the economic event creates a new *economic resource* or modifies some features of an existing *resource*. Each *decrement event* is related to exactly one *economic resource* either by a *use* or by a *consume* relationship. The *consume* relationship means that the economic resource does not exist after the decrement event (the resource is consumed). The *use* relationship means that the economic resource still exists after the decrement event, but some of its features have been modified.

In order to keep track of which *resources* have been used or consumed in order to produce others, the *increment* and *decrement* economic events are related by the *conversion duality* relationship, or in short, *conversion*. The *conversion duality* is an n-ary relationship; in the application model there can be many increment and many decrement events related by a single conversion duality.

Each *economic event* is related to two *economic agents*. The *economic event* in the *conversion process* transfers the control over the *economic resource* from one *agent* to another. Each *event* is related to exactly one *economic agent* by a *provide* relationship, and to exactly one *economic agent* by a *receive* relationship, see Fig. 17. The transfer of control can occur at the beginning, at the end or during the *economic event*. Each *agent* can be related to zero or more *economic events*.
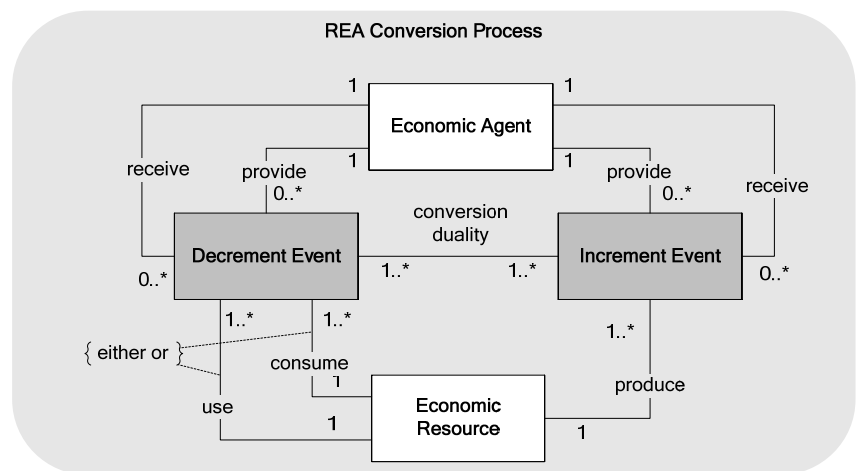

Fig. 17. REA conversion process

In order for a conversion process to add value, the overall increase in value of the resources related to the increment events should be greater than the overall decrease of value related to the decrement events, over the period reflecting the entrepreneurial goals of the enterprise.

The following domain rules apply for any REA application model describing the conversion process.

> Each increment economic event must be related by a conversion duality relationship to a decrement economic event and vice versa.
>
> Each increment event must be related by a produce relationship to an economic resource.
>
> Each decrement event must be related either by a use or by a consume relationship to an economic resource.
>
> Each economic event must be related by both provide and receive relationships to an economic agent.

Resulting Context

The domain rules in this pattern allow application designers to derive and discover new facts from the facts provided by the users of a business application. Therefore, a business application can meet most or all fundamental user needs, even if the user requirements and the designer's knowledge of users' needs are incomplete.

Note that at runtime, for some period of time, there might exist a decrement event that is not paired in conversion duality with an increment event. For example, the oven must be turned on good time before the baking of pizza can start.

4 DESCRIBING EVENTS THAT COULD OR SHOULD HAPPEN

The REA models in the previous section described economic exchanges or conversions that actually occurred. The purpose of these REA models in the previous section is to be able to record every possible scenario that may happen, and therefore these models are as flexible as the reality is. There are logical constraints determining what is physically possible (such as resources must exist before they can be used or consumed), but they are specified as logical constraints, and not as time sequences. What comes first, second, etc. is determined at runtime[7].

However, we would also like a business application to describe what events should, could, or should not occur under certain conditions. For example, trading partners often agree upon transactions that will occur in the future. Likewise, not everything is allowed; law, culture, and internal company rules constrain the economic

---

[7] If we would made a software solution specifying, for example, that Joe always sells pizza first and then customer pays, then if the customer would occasionally pays beforehand (or pays more than expected), then we do not want an application to respond by "Your application has encountered a problem and needs to close. We are sorry for the inconvenience. If you were in the middle of something, the information you were working on might be lost. Please tell your software vendor about this problem. We will treat your business data as confidential and anonymous".

exchanges or conversions that are possible or desirable in any situation. For example, rules might specify what qualification of employees is needed to perform certain operations, or what kind of equipment is needed to transport hazardous materials.

This section describes how to make a software application aware of the rules and constraints, and help users act upon them. By separating the operational level (described in the previous section), and the knowledge[8] level (described in this section), the software application is able to register events that do not conform to the rules, and respond in an appropriate way. For example, the business application could inform the user of the rule violation, advise him on what to do instead, and prohibit him from committing or executing an illegal exchange or conversion.

The *REA COMMITMENT* pattern specifies which events economic agents agreed upon to occur in the future. The central patterns in this section are the *REA CONTRACT PATTERN* and the *REA SCHEDULE PATTERN*, that bind together commitments and terms, which instantiate additional commitments in case the agreed commitments have not been fulfilled. The *POLICY PATTERN*[9] describes certain kinds of business rules, the rules or restrictions that the enterprise wants to apply to the economic events and commitments in which it participates.

4.1 REA COMMITMENT PATTERN

Sales order lines are not economic events; they are promises of economic events

Context
Most economic events do not occur unexpectedly. Economic events are usually scheduled or agreed upon beforehand by economic agents. For example, a sales order line is a promise to sell goods to a customer; the total price is a customers' promise to pay for the goods, and the seller's promise to accept the payment.

Problem
How do we model promises of future economic events?

Forces
Solving this problem requires the resolution of the following forces:

---

[8] The terms operational and knowledge level have been explained and used in (Fowler 1996)

[9] I do not include the POLICY pattern to this paper. Please refer to (Hruby 2006) for details.

- Application designers would like to have a mechanism in the application model specifying details about the promises of economic events. Economic events cannot be used for this, because economic events specify actual increments and decrements of resources, while promises result only in reservations of resources.

- There might be (and usually is) a difference between plans and what actually happens. The users of a business application would like to know whether the economic events occurred as they were promised, and informed about eventual differences.

- If an enterprise promises to give its own resources to its trading partners, users of business application would, most likely, like to know, what resources to expect in return. Conversely, if an enterprise expects to receive some resources, the users of business application would like to know what resources its trading partners expect.

- If an enterprise schedules to the production of resources, users of a business application would like to know what resources it would require to use or consume. Conversely, if an enterprise plans to use or consume resources, the users of a business application would like to know what resources will be produced from them.

- The users of a business application would like to know who should be responsible for the received or produced resources, and who should be responsible for the resources used or consumed during the production or given to other economic agents.

- For each promised exchange, the users of a business application would like to know the trading partners to whom the resources should be transferred, and from whom they should be received.

Solution

Model the promise of the economic event as a *commitment* entity. A *commitment in exchange processes* represents obligations of economic agents to provide or receive rights to economic resources. A *commitment in conversion processes* represents scheduled usage, consumption, or production of economic resources.

Each *commitment* is related to an *economic event* by a *fulfillment* relationship, representing the fact that commitments are fulfilled in the future by one or more economic events executed by the participating economic agents, see Fig. 18. Commitments have usually properties for the *Scheduled Date* or period of the economic event, and the *Scheduled Value* of the event.

The *Scheduled Value* does not need to be expressed as an actual number, but, for example, as a rule. For example, the price of a service can be determined according to actual costs.
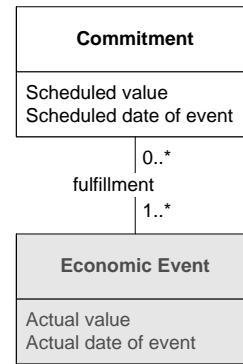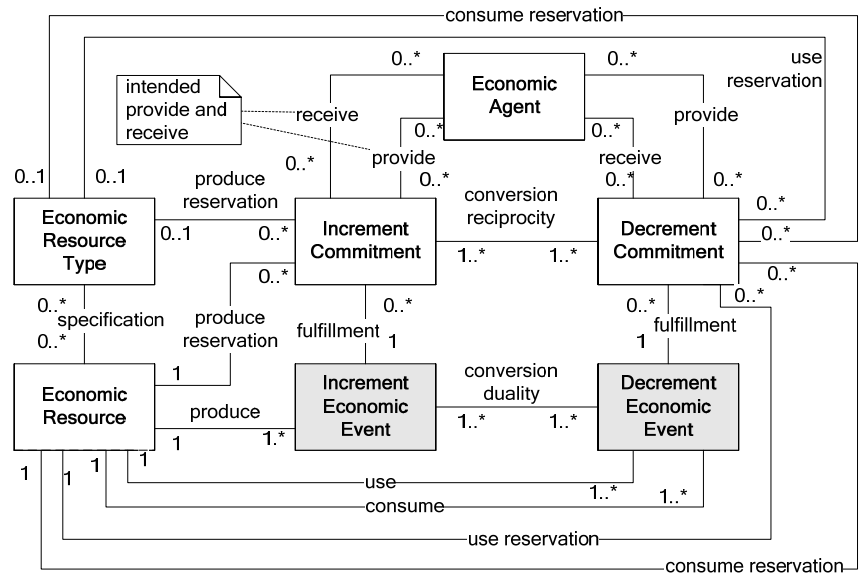
Fig. 18. Commitment and economic event

Each promised exchange and conversion consists of at least two commitments: *increment commitments*, which are expected to increase the value of economic resources, and are fulfilled by increment economic events, and their related *decrement commitments*, which are expected to decrease the value of economic resources, and are fulfilled by decrement economic events. The relationship between increment and decrement commitments identifies which resources are promised to be exchanged or converted to which others, and is called *exchange reciprocity* or *conversion reciprocity.*

Fig. 19 illustrates relationships between commitments, economic events, economic agents, and economic resources in exchange processes. Fig. 20 illustrates the same relationships in conversion processes.
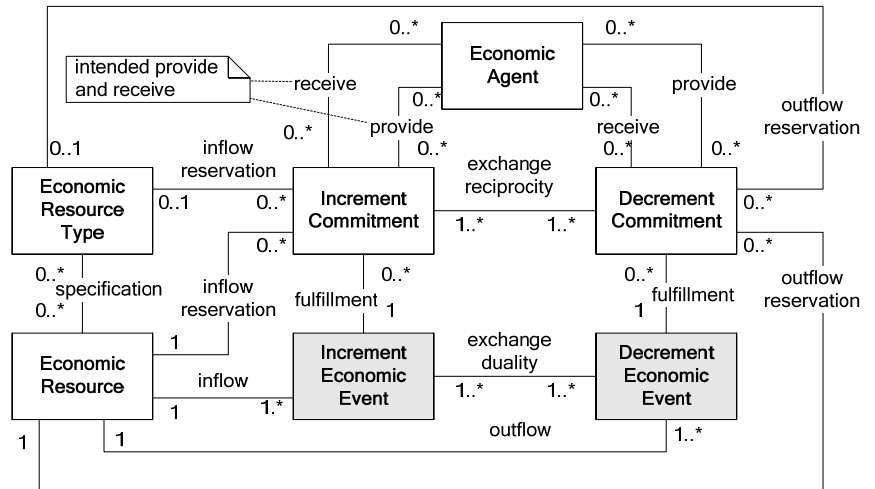
Fig. 19. Relationships of commitments in exchange process

Fig. 20. Relationships of commitments in conversion processes

Domain Rules

The following domain rules apply to any REA application model. As commitments are a mirror image of the economic events at the policy level, the domain rules are similar to the rules for the REA model at operational level, with one addition: commitments must be fulfilled by economic events. These rules can be used to ensure consistency of REA application models.

> Each commitment must be related to a resource, and might (but does not have to) also be related to a resource type.
>
> Each commitment must be related by provide and receive relationships to economic agents.
>
> Each increment commitment must be related by an exchange or conversion reciprocity relationship to a decrement commitment, and vice versa.
>
> Each increment commitment must be related by a fulfillment relationship to at least one increment economic event, and each decrement commitment must be related to at least one decrement economic event.
>
> A commitment that is part of a conversion must be related to the economic event of a conversion process; likewise, a commitment that is part of an exchange must be related to an economic event of an exchange process.

Resulting Context

The reciprocity relationship often has additional functionality that relates together the values of the increment and decrement commitments. For example, in economic exchanges, the reciprocity can calculate the total price (value of the

outflow commitments) based on the line item prices (value of the inflow commitments). The reciprocity might also validate that the cost (value of the outflow commitments) is lower than the price (value of the inflow commitments). The functionality of the reciprocity relationship can vary in different implementations; but the fundamental point is that the increment and decrement commitments are related.

4.2 REA CONTRACT PATTERN

Contracts are statements of intent that regulate the behavior among organizations and individuals. Clauses of a good contract define what should happen in the cases of cancellation and violation of the commitments

Context

Commitments represent the optimistic path of an exchange. For example, a sales order contains commitments to deliver goods and commitments to pay. However, sometimes goods are not delivered as expected and payments arrive late. Partners usually also agree upon what should happen if the initial commitments are unfulfilled.

Problem

How do we specify in the REA model what should happen if the commitments are unfulfilled?

Forces

We need to balance the following forces:

- Commitments specify what economic events should occur. However, in the case in which they do not occur as they should, economic agents usually agree upon what should happen next. The rules specifying what should happen next can be very complex, and keeping track of what should happen, and when, can be cumbersome. Therefore, application developers would like this information present in the business application, so that these rules and actions can be monitored and triggered automatically.

- There are usually several inflow commitments paired through exchange reciprocity with several outflow commitments. These commitments are often considered a unit. Sometimes, it does not make sense to fulfill only some commitments and not to fulfill others, but sometimes this is acceptable. Application designers would like some entity to contain such rules.

- Intended recipients or providers of the resources might be different economic agents than the agents that agree about the exchange.

Solution

If a commitment is unfulfilled, the *terms* of a contract specify additional commitments.

A *Contract* is an entity in the REA application model containing *increment* and *decrement commitments* that promise an exchange of economic resources between economic agents, and *terms*. *Commitments* were discussed in the previous pattern. *Terms* are potential commitments that are instantiated if certain conditions are met. These conditions can be various, such as a commitment not being fulfilled, or a resource being at a certain location. For example, economic agents can agree upon penalties if the commitments are unfulfilled. If the commitments are unfulfilled, the contract will instantiate a new commitment to pay a penalty. The *terms* and *commitments* are the *clauses* of the contract

Every contract must be related to two or more economic agents by a *party* relationship. These agents do not necessarily have to be the provider and recipient of economic resources. The economic agents that are parties in a contract can be different from the economic agents related to the commitments within the same contract, and different from the agents participating in the economic events which fulfill these commitments. For example, a flower shop can deliver flowers to a different person than the one who placed the order, and the flowers will be paid for by a third person, different from the persons who placed the order and received the flowers.
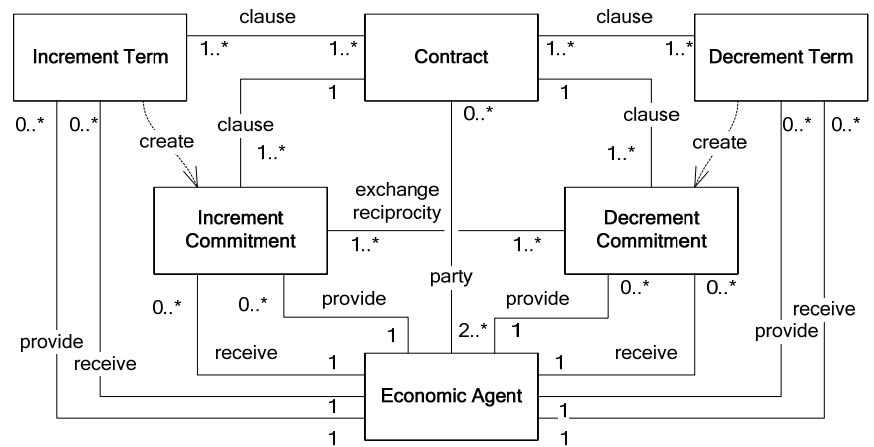

Fig. 21. Contract, commitments and terms

Offer and *Quote* have the same structure as contracts that have not been accepted by all parties in a contract. Economic agents negotiate the content of the commitments and terms, and when they agree upon commitments and terms, the quote or offer becomes a contract that binds the agents that are parties in the contract. There is usually certain period of negotiations and draft versions from when the offers and quotes are created and until the contracts are accepted by both contracting parties.
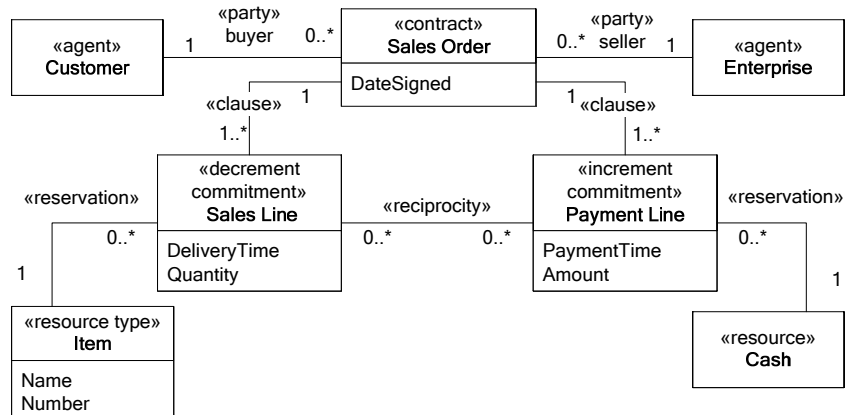
Examples

Fig. 22 illustrates a business document for a simple sales order without delivery and payment terms. The REA application model for this sales order is illustrated in Fig. 23. The *Sales Order* contains two *Sales Lines* specifying the goods; the sales line entity corresponds to the line item in Fig. 23. The *Payment Line* specifies the price (i.e. expected amount of received cash); the entity *Payment Line* corresponds to the *Total* line in Fig. 23.

Sales Order			
Enterprise: **Joe's Pizzeria**		Date: 11 May, 9:15	
Customer: **Addy**			
Number	Item	Quantity	Delivery Time
6128	Pizza Margherita	2 units	11 May, 18:00
8694	Cola 0.5l	1 unit	11 May, 18:00
	Total	**21,00 USD**	11 May, 18:30

Fig. 22. A sales order is an example of a contract

Fig. 23. The REA application model of a sales order

Fig. 24 illustrates an instance of this Sales Order (an actual sales order that conforms to the application model from Fig. 23; the data of this sales order corresponds to that in the example in Fig. 22.

Note that the REA model does not specify how to calculate *Total*, e.g. how the amount of 21 USD is related to two units of pizza and 0,5l of Cola. The calculation rules may range from a simple sum of the unit prices of the pizza and cola, to the complex rules taking into the account the identity of the customer, date, time, and volume of the sale. The fact that price calculation vary from one software application to another, is the reason why the price calculation is not part of the REA model. The REA model formulates the fundamental principles common to all business applications. The section on Service Patterns shows how to extend the REA skeleton by application specific functionality.
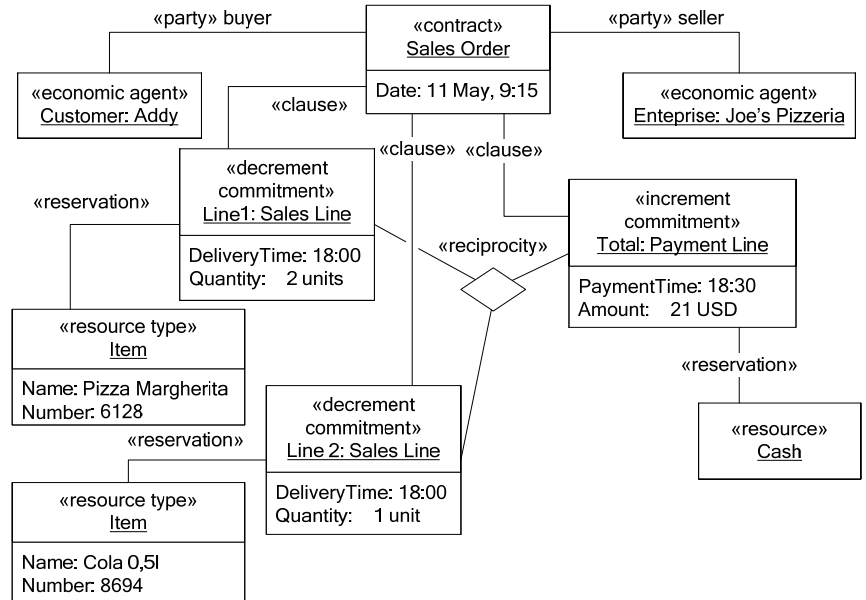
Fig. 24. An instance of the REA application model of a sales order

Fig. 25 illustrates a more complicated example of a sales order with shipment and payment terms. For example, Joe' Pizzeria and Addy agree that Joe's Pizzeria will sell five units of Pizza Margherita to Addy on Tuesday, and Addy will pay for them on Friday. If Joe's Pizzeria does not ship on Tuesday, Joe's Pizzeria pays a 20 USD penalty to Addy on Friday. If Addy does not pay on Friday, he pays a 30 USD penalty to Joe's Pizzeria the following Monday. The informally sketched properties of the terms and commitments can be implemented as *DUE DATE PATTERN* and *VALUE PATTERN*; see section 6, Extending the REA Model Using Services.
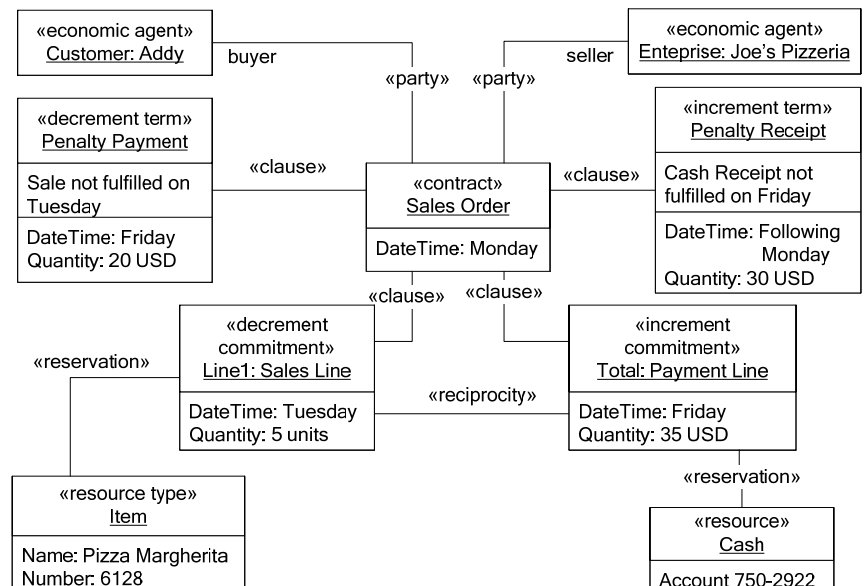

Fig. 25. Simple contract with shipment and payment terms

If Joe's Pizzeria does not deliver 5 units of Pizza Margherita on Tuesday, the contract instantiates the penalty, and the model between Thursday and Friday looks as in Fig. 26.

After the Penalty Payment commitment has been instantiated, all commitments still need to be fulfilled by economic events. According to this contract, the Joe's Pizzeria still has to ship and the agent Addy has to pay, even in the case in which Joe's Pizzeria does not ship at all. A better contract might specify that the payment should occur within a certain time period from the shipment.
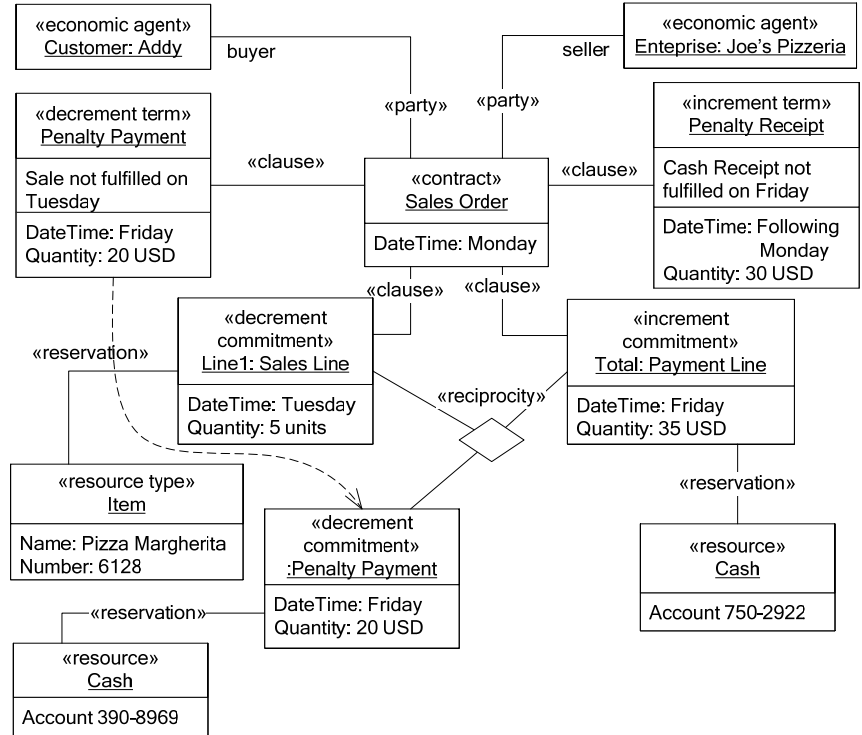
Fig. 26. Simple contract after one of the terms' conditions has been met

Resulting Context
The precise specification of commercial contracts is a subject of intensive research. Simon Peyton-Jones, Jean-Marc Eber, and Julian Seward have developed a functional language for financial contracts; this language does not have an REA concept of reciprocity (Peyton-Jones, Eber 2003). A language for REA-compatible contracts is being developed by Fritz Henglein and his students (Henglein 2005).

4.3 REA SCHEDULE PATTERN

Schedule is a series of things to be done or of events to occur at or during a particular time or period

Context

Production processes usually do not occur spontaneously; a rational company schedules the production and usage of its resources that should take place in the future. However, production sometimes does not occur as planned because of unexpected circumstances. A rational company would like to mitigate risks and determine additional factors that should occur if the originally planned operation does not occur as expected. Making a plan is a way to minimize the risks of missing some resources in the middle of a production. The purpose of the plan is to make sure that for all processes the needed resources are identified, as well as when they will be needed.

Problem

How do we specify conversion processes that should occur in the future?

Forces

The following forces influence the solution:

- If use, consume, and produce economic events do not occur as commitments specify, the enterprise would like to have an alternative plan to mitigate the consequences. Application developers would like this information present in the business application.

- A conversion process usually consists of several use, consume, and produce economic events that have various, often complex dependencies on each other. If some of these events do not occur as committed, the mitigation plan depends on a combination of the values of the economic events. The application model should contain an entity containing such dependencies.

- The economic agents that are responsible for the overall conversion process can be different from the agents that control the economic resources.

Solution

A schedule is a collection of increment and decrement commitments in conversion processes and mitigation plans. Mitigation plans instantiate additional commitments under certain conditions, typically if some of the original commitments are unfulfilled, see Fig. 27. Unlike invoking penalties in the contracts, instantiating

commitments from mitigation plans is usually not an automated task, and it requires the assistance of the users of business applications.

A schedule is related by a party relationship to the economic agents that are responsible for the schedule. The agents that are related to the schedule can be different from the agents that are related to the commitments. There are usually two agents related to the schedule. One of the agents sets the requirements of what should be done (representing a client in the planning process), and another agent is responsible for the actual conversion, (representing the supplier in the planning process).
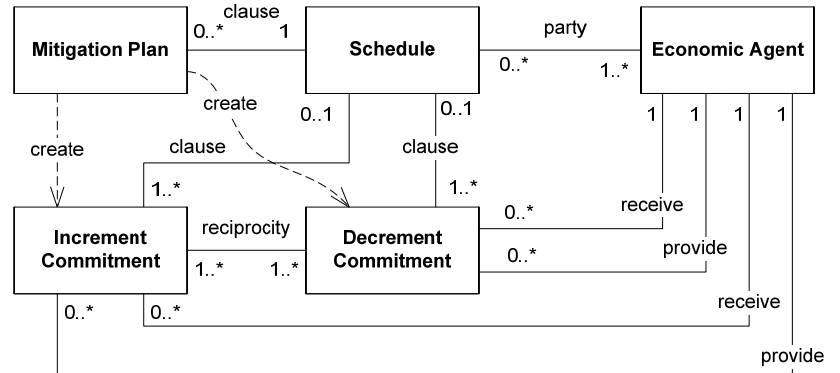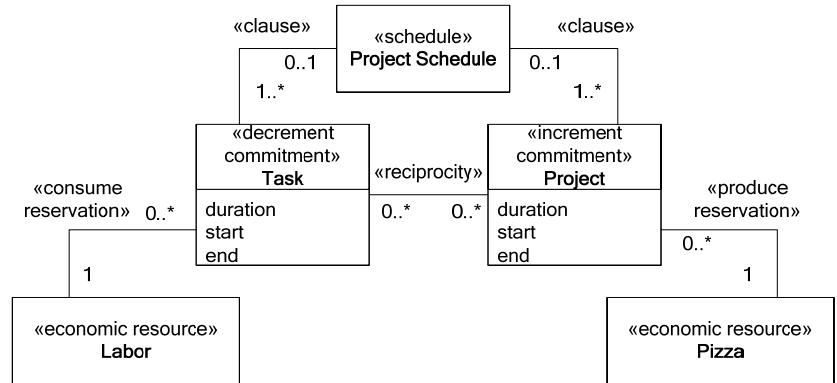

Fig. 27. Schedule

Example

The example in Fig. 28 illustrates a simple schedule of a project Produce Pizza, assigned to Tom, Susie, and Mike. Project Produce Pizza is an increment commitment, and the consumption of the labor of Tom, Susie, and Mike are decrement commitments.

| ID	Task	Resources	Duration	11 February 2005											
				7	8	9	10	11	12	13	14	15	16	17	18
1	Produce Pizza		10h												
2	Dough	Tom	1h												
3	Toppings	Susie	3h												
4	Baking	Mike	4h												

Fig. 28. A simple schedule

The REA application model corresponding to the diagram in Fig. 28 is illustrated in Fig. 29. The schedule Project Schedule has an increment commitment Project, which reserves (expects) the economic resource Pizza. The decrement commitment Task reserves consumption of the economic resource Labor. The properties start, finish, and duration can be implemented as *DUE DATE SERVICE PATTERN*; see (Hruby et al. 2006).

Fig. 29. REA application model for simple schedule

Fig. 30 illustrates an instance of the REA application model from Fig. 29 that corresponds to the example in Fig. 28.
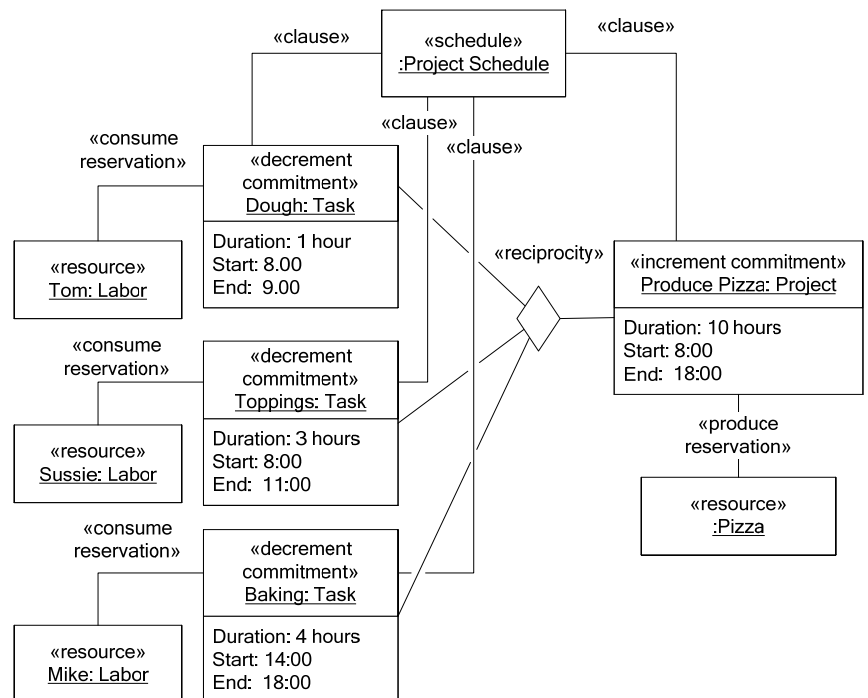
Fig. 30. An instance of the REA application model of a schedule

There are many examples where the detailed schedule means success or failure for the whole company. In just-in-time production, the resources are delivered exactly when they are needed. Delivery too early would mean a need for storage and late delivery can stall the production.

5 CAPITALISM FOR SOFTWARE ENGINEERS

Capitalism generally refers to an economic system in which the means of production are privately owned; individuals and groups of individuals have the right to trade capital goods, labor, land and money; and pricing is determined through the operation of a free market.

The REA modeling framework expresses these principles in terms of software design. Therefore, REA software is applicable more or less to all environments for which the aforementioned economic principles apply. The simplicity of the model is paradoxically a reason why it is sometimes difficult to understand.

For example, people sometimes expect that in the exchange process, the total value of the resource given away should match the value of the resources received, much as an order total should be calculated by summing up the prices of the line items. This is a common practice, but not true in all cases.

Each resource that is subject to exchange has a different value for the economic agents participating in the exchange. The REA model specifies nothing more than an economic exchange can occur only if both economic agents perceive the value of the received economic resources higher than the value of the given resources; otherwise, they will not exchange them. For each agent, the value of the received resources is higher[10] than the value of the given resources, in the perspective of the entrepreneurial goals of each agent.

For some modelers it is difficult to specify what an enterprise receives in return for giving donations – for example, why CSC sponsors a cycling team. From the REA perspective, CSC will receive more value in return for selling its services, taking into the account the whole lifetime of the enterprise, than if it would if not support the cycling team. Otherwise it would not do it.

In the conversion processes, such as software development or pizza production, the overall incremented value of the produced resources (considering the enterprise's entrepreneurial goals) is also higher than the overall decremented value of the consumed or used resources, but it does not mean that every actual increment event must increase the value; the increment events increase the overall value of the resources over the period reflecting the entrepreneurial goals of the enterprise. A specific production run can be unsuccessful and sometimes the value of the resources is decreased. However, on average the process must add value; otherwise, a rational enterprise would not perform this process.

---

[10] Some economists argue that in the conditions of ideal competition, these values are very close. However, a precise statement (needed for software design) is that for each agent, the received value is higher than the given value, but we are unable to determine how much.

6 EXTENDING THE REA MODEL USING SERVICES

The previous section, What Is the REA Model, discussed the structure of a business application, which conforms to the laws of the business domain, consisting of REA entities and their relationships. To build a useful business application, this structure is only one of the things an application developer has to determine. Users of business applications usually require additional functionality, such as serial numbers, accounts, price calculations, and conversions between units of measure. This functionality is essential in some applications, but it might not be required in others. All depends on the users of a business application, actual configuration of an application, and the common practices in their businesses.

In this part, Extending the REA Model Using Services, I describe how the REA model can be extended to support specific functionality that originates in specific user requirements, and which might change in time, often together with changing technologies.

6.1 BEHAVIOR MAY NOT BE LOCALIZABLE INTO REA ENTITIES

Units of functionality that extend the REA model are usually not localizable into a single REA entity. An example is illustrated in Fig. 31. This example shows the economic resource *Vehicle*, which belongs to the *Vehicle Category*[11], and is used in the economic events *Trip*.
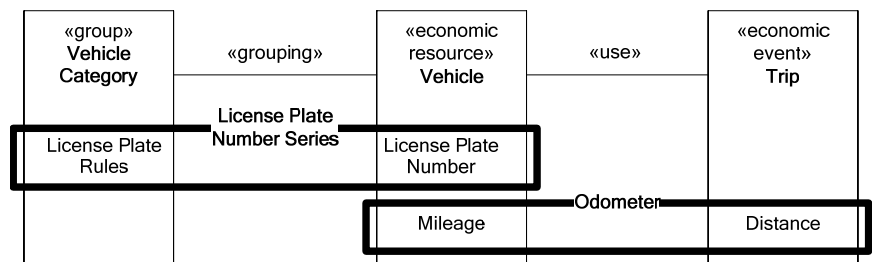

Fig. 31. Service patterns often crosscut REA entities

A *License Plate Number* of a vehicle is an attribute of the economic resource *Vehicle*. The *License Plate Number* is usually not a random number. It is constructed using a *License Plate Rules*, which is a property of *Vehicle Category* (for example, numbers of police cars, military cars, and diplomatic cars are constructed using different rules than numbers of other cars). The property *License Plate Rules* contains rules specifying the uniqueness of the *License Plate Number*, its format, its dependency on previous numbers or other attributes, and so on. Therefore, the unit of functionality of a *License Plate Number Series* is present on two REA entities, the resource and the resource group, and the number is constructed by mutual collaboration between the part that resides on the resource and the part that resides on the group.

Likewise, a *Mileage* of a *Vehicle* is calculated as the aggregated number of the trip *Distances* the vehicle traveled. As *Trip* is an economic event, the *Odometer* is a

---

[11] Category is an REA entity out of scope of this paper. Please see (Hruby 2006) for details.

unit of functionality present on two REA entities, the economic resource and the economic event.

It is still useful to think about a *License Plate Number Series,* and about an *Odometer* as single units of functionality, but these units span several REA entities.

6.1.1 ASPECT-ORIENTED PROGRAMMING

Aspect-oriented programming is one of the mechanisms for describing the crosscutting features and manipulating them as modular units. Aspect-oriented programming is based on the ideas of Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, and John Irwin, (Kiczales 1996). This group at the Palo Alto Research Center, a subsidiary of Xerox Corporation, developed a general purpose aspect-oriented language called AspectJ, an extension of the Java programming language with aspect-oriented features. Many other research centers have developed other aspect-oriented languages, both general purpose and specific to a certain domain.

Aspect-oriented languages, such as AspectJ, express the structure of a software application in the form of code in the programming language, and the crosscutting concerns, or aspects, are also expressed as code. During compilation, both the application code and the aspect code are combined together in a process called weaving; see Fig. 32.
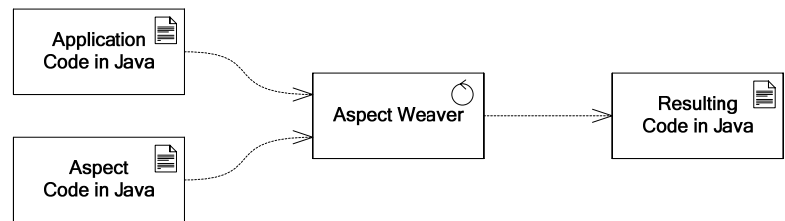
Fig. 32. Aspect-oriented programming at the code level (not framework-based)

Keeping in mind requirements such as extensibility and configurability, a disadvantage of such programming languages is that the code has to be weaved (which means recompiled) every time the functionality of an application (expressed as application code or aspect code) changes. The consequence is that upgrading an application is complicated and expensive.

Furthermore, since some or all functionality of an object is provided in the aspect code, it is impossible for the weaver to guarantee a system-wide quality for an application, because the weaver has no way of knowing what the aspect code does.

6.1.2 SERVICE-ORIENTED APPROACH

To satisfy the requirements for extensibility, configurability, and upgradeability, I use the service-oriented concepts to model the cross-cutting concerns. Every service is represented at two levels of abstraction, the *Service Type level* and the *Model Configuration level.*

The *Service Type* level specifies the types of the services, and metadata that can be applied to the services in the application model. This level encapsulates the business logic of the service, and specifies the configuration properties, which can be set by application developers.

The *Model Configuration* level specifies the runtime attributes that can be set by the users of business applications or automatically by the system. The model configuration level also specifies which services are configured on which REA entities; shown by dashed lines in figures in this section.

Advantage of a system with explicitly modeled service types is that software business applications are much easier to configure, customize and upgrade than if the services were to be represented only as code in a programming language.

Configuration of software business applications using the service patterns is basically reduced to creating an REA model, setting the configuration parameters of services, and specifying which services extend which objects. This can be done using a specialized designer without writing any code in a programming language.

Furthermore, software applications are easy to upgrade, because all application logic is encapsulated in the elements at the service type level, and it can be extended independently of the configured application model. The upgrade of the software application basically means replacing the components at the service type level with components with upgraded functionality. The service developer designs the interface or contract (the configuration properties and the corresponding behavior) that the components at the service type level expose. If the upgraded components support the old interface, the software applications can be upgraded without reweaving or recompiling the application.

Even if the upgraded services are not backwards compatible (backward compatibility is considered anti-pattern by some practitioners), it is possible to write an upgrade script that modifies the configured applications to support the upgraded services.

Quality of the software applications is easier to control, as all functionality of business applications is encapsulated in the service types, and is therefore tested by service developers. The service developers have full control over what application developers may do with their service components. In other words, providing application developers a domain-specific modeling language (specified by service interfaces) reduces the number of errors the application developers can make, compared to the situation in which the application developers write code in a general programming language.

6.1.3 THERE IS NO COMPLETE LIST OF SERVICE PATTERNS

While with the REA model our aim was to find the minimal, yet complete set of abstractions covering the business domain, this is not possible with service patterns. Users of business applications will always need new features, and service patterns provide a mechanism to add new features to a business application without changing its fundamental structure.

There are service patterns waiting to be discovered. This section describes the patterns I came across in developing business solutions, but it is not a complete list of all patterns that might be needed in any line of business. As the REA structural patterns define more or less a complete set of concepts, if application developers identify user requirements for new functionality, they would likely be either new service patterns which crosscut the REA entities or features in a domain other than the business domain.

6.2 IDENTIFICATION SERVICE PATTERN

Barcode is a machine readable strip for automatic identification of items, by means of printed bars of different widths

Context

People refer to real or imaginary things by their names. We name things to identify them, so we can refer to them by their names and not just point to them and say "this!". By naming, we give things identities, but in real life they are not often unique. Many things have more than one name, and sometimes a single name can refer to different things, which is fine as long as everyone who uses that name knows what thing it refers to. In business, people use serial numbers, production numbers, civil registration numbers, and names.

Problem

How do we specify the identity of things represented by REA entities?

Forces

The solution needs to balance the following forces:

- An identity is a given feature; it is not an intrinsic part of the objects and things[12]. Therefore, an REA application model must specify whether there is a business reason requiring REA entities to have a distinct identity, and how that identity is modeled. We could omit modeling identity of an entity, but then we could distinguish different instances of this entity only by the values of their attributes.

- Users of business applications do not necessarily require that all REA entities have an explicit identifier. For example, users of business applications might not be interested in managing the identifiers of sales order lines.

- Some identifiers are unique in the universe, such as the GUID (Global Unique Identifier); some are not unique, such as the first name and last name of a

---

[12] The Ship of Theseus is a paradox that raises the question of whether an object, which has had all its component parts replaced, remains fundamentally the same. Consequently, identity is context-dependent; sometimes makes sense to regard an object's identity as the same for a particular purpose even if it might be different for some other purpose.

person. Some identifiers are unique within a certain group, such as a serial number, which is unique within the group of entities that belong to the same number series.

- There are specific rules on how to construct identifiers. For example, the ISBN (International Standard Book Number) or the numbers of major credit cards are constructed in a way that enables verifying, using a simple calculation algorithm, whether the number is valid.

Solution

The *Identification Service Pattern* can be used in situations in which application developers want to specify the identity of REA entities. In the REA application model, the *Identification Service* consists of *Identifier* that represents the name or number of an REA entity, and *Identifier Setup* specifies the rules for creating the Identifiers.

The *Identifier Setup* is often configured on *group* of REA entities that share the same rules for creating identifiers, for example, on a group that belongs to the same number series. The *Identifier* can be configured on any REA entity that needs to be identifiable, including the groups. As not all REA entities are parts of some group, the *Identifier Setup* is often omitted from the model, or is implicit in a software application, for example, as a system table.
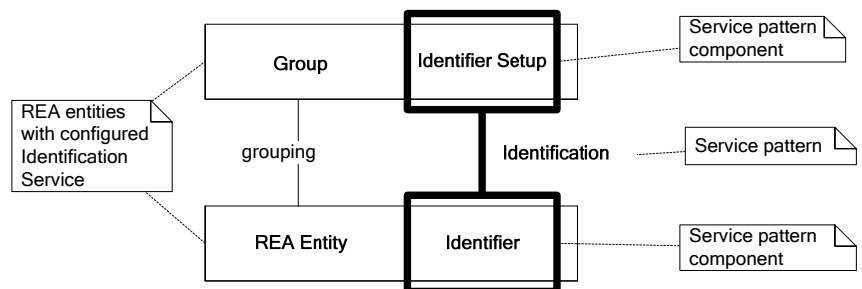

Fig. 33. Identification service in the application model

In all figures in this section, the Service Elements are shown as rectangles with thick line; their runtime properties are shown similarly, as UML attributes. Properties of the service component types (i.e. the properties whose values are set at the service type level) are shown in the name compartment of the service. Values of the properties of the service type are shown as text close to the line connecting the service elements, similarly as UML attributes; for example, 'Mandatory = yes'.

Solution Details

The *service type level* encapsulates the business logic of the service and configuration parameters. At the service type level, the *Identification Type* defines the *Name* of the type of identification, as well as other attributes. *AutoNumber* is a Boolean function that can be set on or off to indicate whether the *Identifier* can be automatically generated by the identification service or not; automatically generated number is often referred to as a number series. *Unique* is a Boolean function that can be set on or off to indicate whether or not the *Identifier* is required to be unique

at runtime. *Mandatory* is a Boolean function that can be set on or off to indicate if the *Identifier* must be defined at runtime or can be undefined.

The *Identification Type Service* has two elements, *Identifier Type* and *Identifier Setup Type*. These elements contain business logic for interpreting the *ID rules*, and logic for creating and validating *Identifiers*. They do not have any configuration parameters; just serve as metadata for the *Identifier* and the *Identifier Type* at the application level.

The rules for creating new *Identifiers* can vary from simple series with linear increments to rules that allow for validity checks of the identification strings, such as credit card numbers. Legislation in some countries requires that numbers of some business documents consecutive, without gaps, which imposes an extra requirement on how the number is constructed. If an REA entity has been created by omission and deleted after another REA entity of the same series has been created, the *ID Rule* must be able to identify the gap in the series and reuse the number of the deleted document.
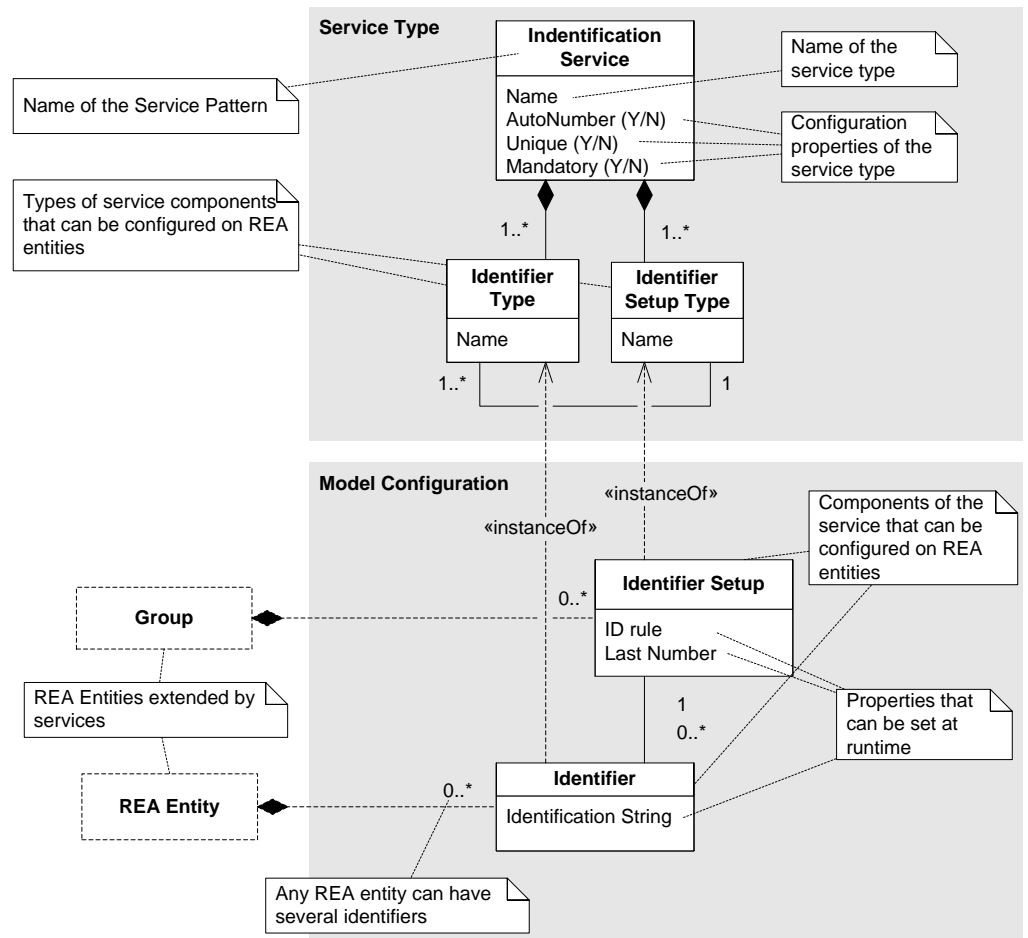
Fig. 34. Details of the identification service pattern

The *model configuration level* specifies the runtime attributes that can be set by the users of the business application, or automatically. At the application model level,

the *Identifier* is configured on the REA entity that should have some form of identity. The *Identifier* contains the *ID String*, which provides an identity to each REA entity instance.

The *Identifier Setup* is usually configured on a group[13] of REA entities that share the same ID rule for creating or validating an Identifier. The *ID Rule* determines how the identification strings are created (users of business applications often use combinations of letters and numbers). The *ID Rule* can also be used for validating the identification strings entered manually by the users of the business application. If the *Identification Type* service is an *AutoNumber*, the *Identifier Type* also has an attribute *Last ID*, which defines the last used identification string in the series.

Examples

Sales Order Number is an identification that is an auto-number, is unique, and is mandatory. As the *Sales Order Number* is an auto-number, the *Identifier Setup* element contains the attribute *Last Number*.
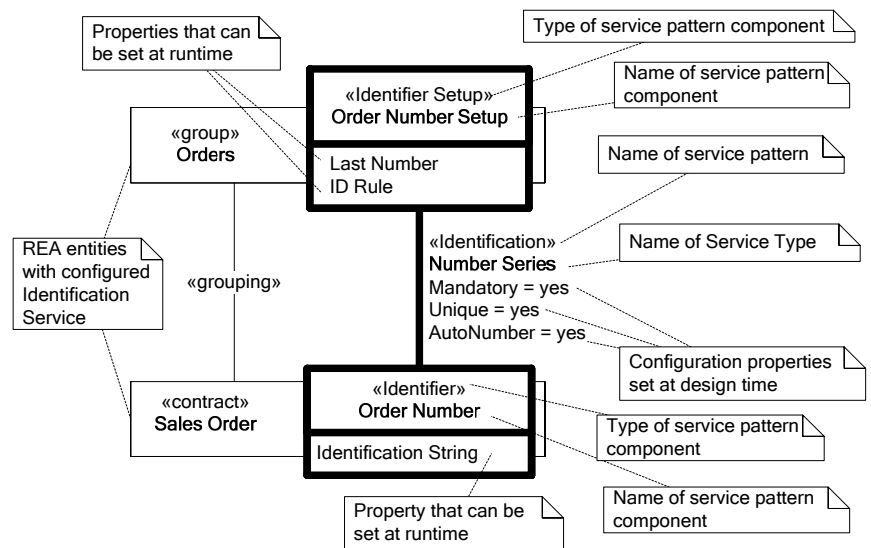
Fig. 35. Sales order number

The *Social Security Number (SSN)* of a person is an identification that is not an *auto-number*, is *unique*, and is *not mandatory*. The *Identifier Setup* has the name *SSN Numbering Scheme*, and contains an *ID Rule* that determines how the social security number is constructed or verified. The *Identifier* has the name *Social Security Number*, and its *ID String* at runtime contains the social security number.

---

[13] Group is an REA entity out of scope of this paper. Groups represent heterogeneous collections of REA entities. Please see (Hruby 2006) for details.
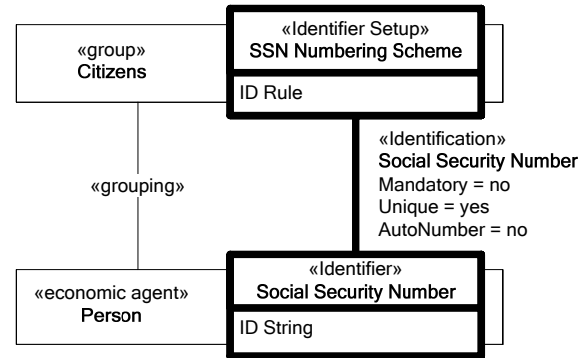
Fig. 36. Social Security Number

Resulting Context

Sometimes, users of business applications use *phone number*, *e-mail address*, or *Internet address* as identifiers of their trading partners. These numbers and addresses have multiple and different semantics. Phone number can also be used as a contact address, e-mail address as a contact address and destination location (for sending electronic documents and products), and Internet address as a description of the trading partner. In such cases, different services will contain or refer to the same data (both identification and notification will contain or refer to the same phone number).

There are several international standards specifying Identification Strings and ID rules for economic resources and economic agents in various lines of business. Examples are European Article Numbering (EAN) for industrial products, International Standard Book Number (ISBN) for books, International Standard Serial Number (ISSN) for periodicals, and International Standard Music Number (ISMN) for printed music publications. For companies, the Data Universal Numbering System (DUNS) is used. References to these standards can be found, for example, in (Arlow, Neustadt 2003).

6.3 LOCATION SERVICE PATTERN
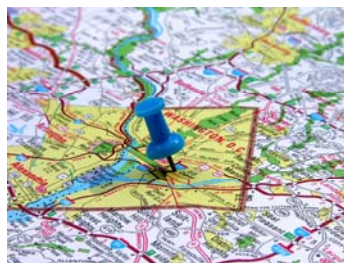
Location is a point in space

Context

Most economic events take place in time and space. For some economic events, the location is an essential attribute characterizing them. Shipment, for example, is an economic event in which an economic resource is moved from one location to another. Users of business application are interested not only in departure and

destination, but often also in the actual location of the economic resource during the economic event.

Problem

How do we specify where the economic events occur?

Forces

We need to balance the following forces when creating the model:

- Economic resources that are physical in nature are usually located at specific places in the world. Users of business applications would like to know where a resource is.

- Information modeled as an REA economic resource also has location. Information is always stored on a medium that has a location, and information can be transferred from medium to another.

- Economic events contain historical information about changes of features of economic resources or transfers of rights to these resources. These changes and transfers occur both in time and space.

- Economic resources can change their locations as a result of economic events or by forces outside of the scope of the application model. If an economic event changes the location of the resource, users of business applications would like to plan, monitor, and control changes of locations of the resources.

Solution

In the REA application model, the Location is a service consisting of Position and Route, see Fig. 37. Position specifies the actual position, and Route represents the changes of the Position. Position is usually configured on an economic resource; and Route can be configured on a commitment, which specifies the indented route, or on an economic event, which specifies the actual route.
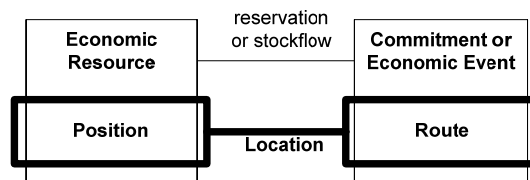

Fig. 37. Location service pattern

Solution Details

The service type level encapsulates the business logic of the service and configuration parameters. At the service type level, the Location Service defines the Name of the location service. The Location Service consists of Position Type and Route Type, both having properties defining their names. The Route Type has a method DisplayMap() that displays the actual route and navigation instructions.

The application model level specifies the runtime properties of the service elements. At the application model level, the Position element has an attribute Actual Position which contains the actual position of the resource. The Route

element specifies a route segment that represents a change in the resource location. The Route element contains the properties Origin, Destination, and Distance.
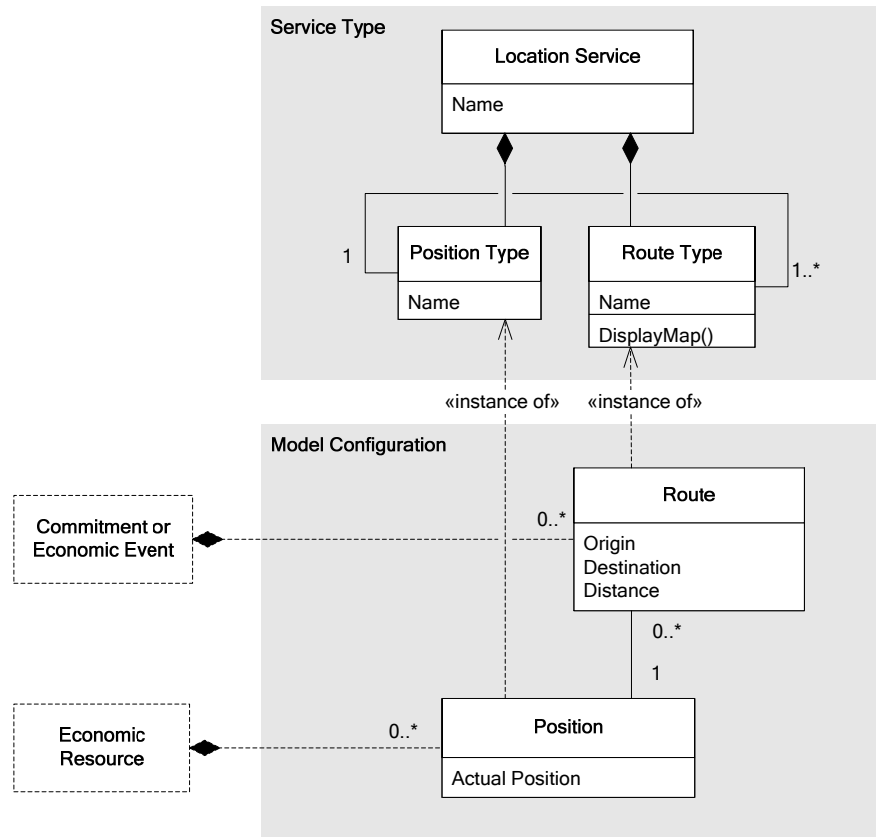

Fig. 38. Details of the location service pattern

Examples

The example in Fig. 39 illustrates a model of the location pattern configured as a Shipment Address. The route segment element is configured on the Shipment economic event; the Destination property represents the final address. The Position element is configured on the Item; the Location property represents the actual location of the Item.
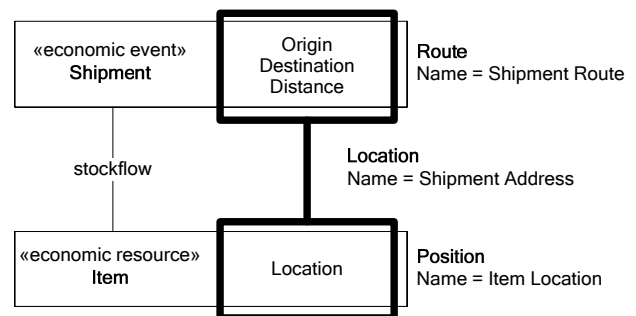
Fig. 39. Shipment address

The location pattern can be used to model an itinerary that consists of several route segments. The travel itinerary is essentially a schedule. There are several ways to construct the REA model for an itinerary, depending on the level of detail of information the users of business application would like to plan, monitor, and control. I will present one possible design. The whole route is represented as an increment commitment, Transport, and its Route service element contains the origin and final destination of the economic resource Cargo. The decrement commitment, Cargo on Carrier, represents the time interval on which the Cargo is loaded onto a specific carrier; it is a decrement commitment because when Cargo is on a vehicle, its possible use for other purposes is limited. The commitment Cargo on Vehicle has a Route service element called Segment, representing a segment of the scheduled transport. At runtime, there can be several instances of the Cargo on Carrier commitment, for example, if cargo is transported using several vehicles or means of transportation.

The other decrement commitment, Vehicle Use, has also a Route service element. The Origin and Destination of the Vehicle Use element and the Origin and Destination of the Cargo on Carrier element can be different, for example, if a vehicle drives unloaded to the loading destination, and then transports Cargo and, again, drives back unloaded. The cost of using unloaded vehicle should be reflected in the cost of the transport, which is what the model does.

An itinerary usually also contains information about time, such as when cargo has been loaded, unloaded, and reached the final destination. This can be modeled using the *DUE DATE SERVICE PATTERN* on commitments and the *POSTING SERVICE PATTERN* on economic events.

The location service pattern can also be configured on economic events instead of commitments; the business application would then monitor the actual movement of Cargo.
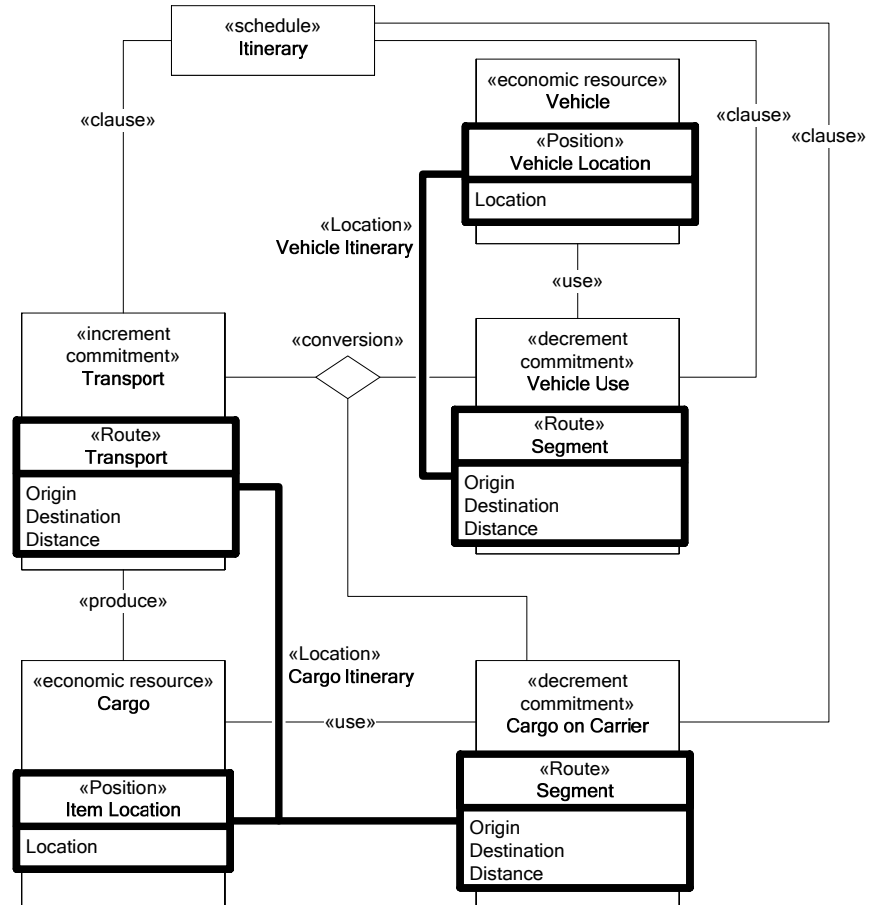
Fig. 40. Travel itineraries for cargo and vehicle

Resulting Context

How do we determine the shipping address of the customer? Some business applications store a shipping address (as well as the billing address, and many other addresses) as attributes of a customer. These addresses are then used as default addresses for shipments, invoices, etc. Users of business applications have the option to overwrite these addresses in case a customer wishes to use a different address than his default address.

The described solution does not have a fixed default customer address. Economic events contain all relevant historical information about business relationship between the enterprise and the customer, and commitments that the enterprise gave to the customer. The economic events and commitments also specify the shipping, billing, and other addresses the customer has used in the past. The address for the next shipment can then be determined by browsing the list of economic events. The customer may then choose one of the existing destinations, a new one, or one that the business application can suggest, for example, the destination of the last shipment, as a default address. This solution is more flexible than that of a fixed default address as a property of the customer entity because it develops automatically as the enterprise's information about the customer develops.

6.4 NOTIFICATION SERVICE PATTERN

SMS (Short Message Service) is a text message to be sent and received to a mobile phone via the network operator

Context

Various users of business applications should often be notified when certain events occur, or when certain conditions become true. For example, both customer and bank personnel might be interested in being notified when the customer account has been overdrawn. Business applications can be configured to create and send notifications automatically.

Problem

How do we notify users of business applications about changes in the REA entities?

Forces

Several forces arise when designing the solution:

▪ There are different ways to contact users of business applications. The notification can range from a message box window on a computer screen to sending a letter to a specified address.

▪ Different users of business applications can be contacted in different ways. Some users can be contacted in multiple ways. The method of notification can vary, depending upon the user and upon the kind of notification.

▪ Different users are interested in different information resulting from the same change.

Solution

Notification is a specific unit of functionality that encapsulates the mechanism for notifying users of business applications. A notification service pattern consists of Address, containing the way to contact the economic agent, and Message, containing the information forwarded to the agent, as well as the logic determining when the agent is notified.
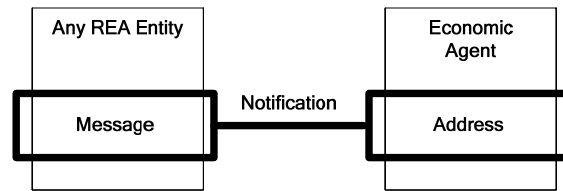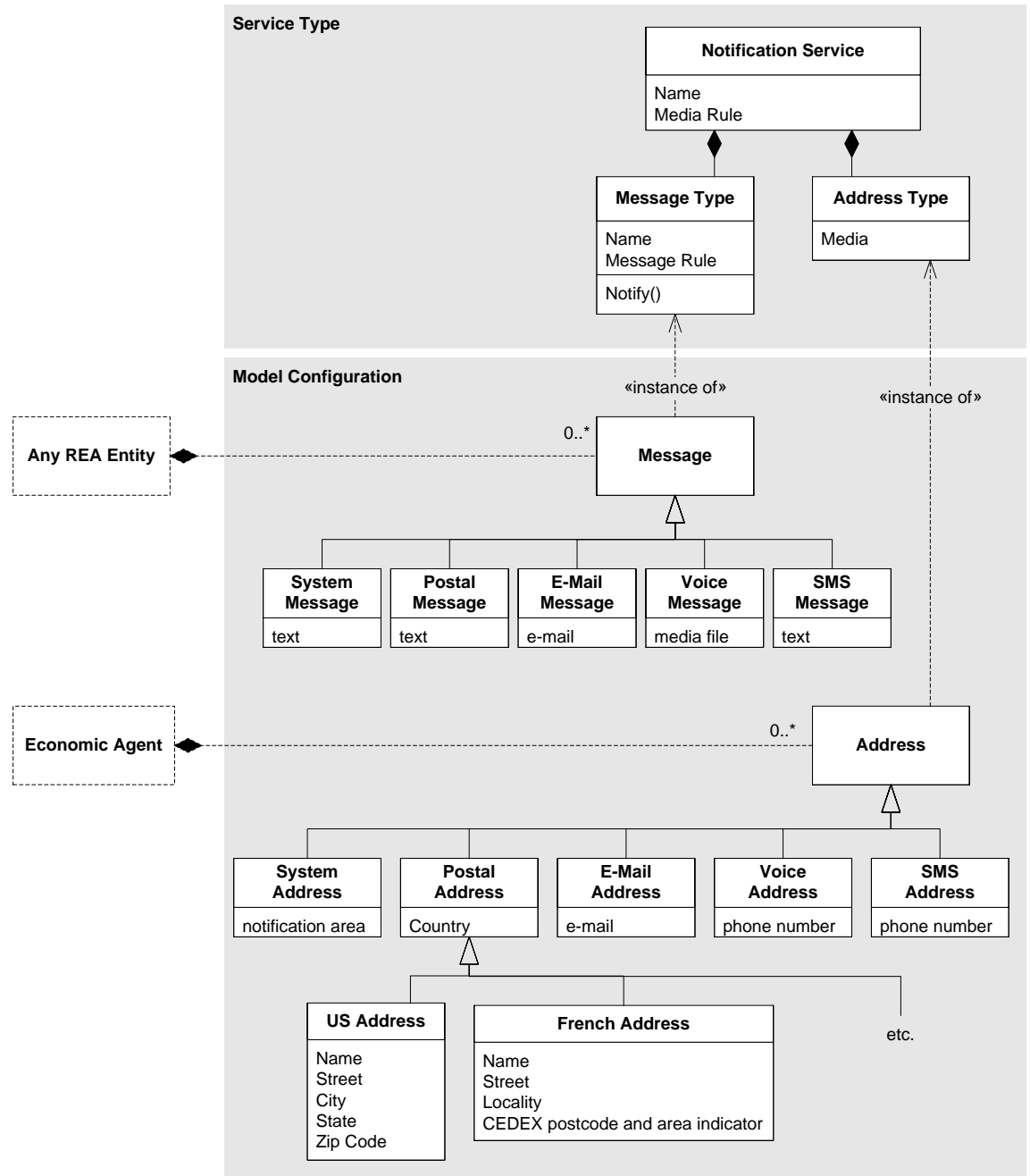
Fig. 41. Notification service pattern

Fig. 42. Details of the notification service pattern

Solution Details

At the service type level, Notification Type contains the Name of the notification, and encapsulates the business logic of forwarding messages to specific addresses. The Media Rule defines which Media the specific Address is allowed to contain, and hence determines which attributes a specific Address Type contains (for example, street, city, and zip code for postal address), and consequently also which message types can be delivered to which kinds of addresses; hence, the Media Rule. Examples of Media are postal address, e-mail address, and SMS address.

The *Message Type* has the responsibility of creating a message. *Name* specifies the name of the message type. The *Message Rule* attribute specifies how the message will be created. The simplest approach is to use a predefined message for each message type; a more complex approach is to create a message at runtime by composing it from predefined information and relevant data available. When the Notify() method is called, the message is created and the user notified.

At the model configuration level, Message can be configured on any REA entity, and represents a message that can be sent to an Address. Message can be one of the listed examples of messages (System, Postal, E-mail, Voice, SMS, and so on). Address is usually configured on an economic agent, and can be one of the listed examples of addresses. Each address contains different elements and rules. The business logic at the service type level determines which message types can be delivered on which address types.
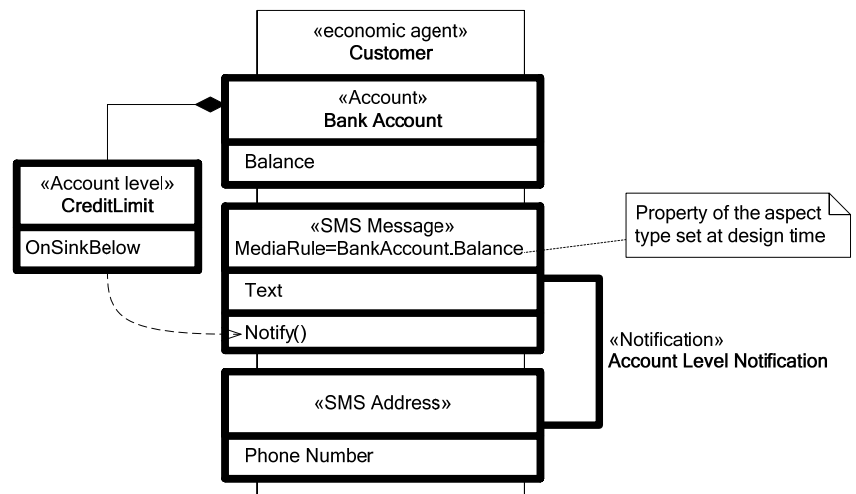

Fig. 43. Notification on account level event

Examples

Fig. 43 shows an example of a Customer economic agent that is notified when its Account level (a part of the Account Type element, see *ACCOUNT SERVICE PATTERN* in Hruby et al. 2006) sinks below its credit limit. Customer is configured with the Notification service, where both the Message and the Address elements are configured at the Customer entity. The OnSinkBelow event of the Account Level causes the Notify() message of SMS Message element to send an SMS message with the Balance of the Bank Account service element as Text

A mobile phone operator, T-Mobile, in some countries sends a voice and an SMS message to its customers every time a customer receives a message in his voice mail. Fig. 44 shows how this functionality could be implemented using the notification service pattern.
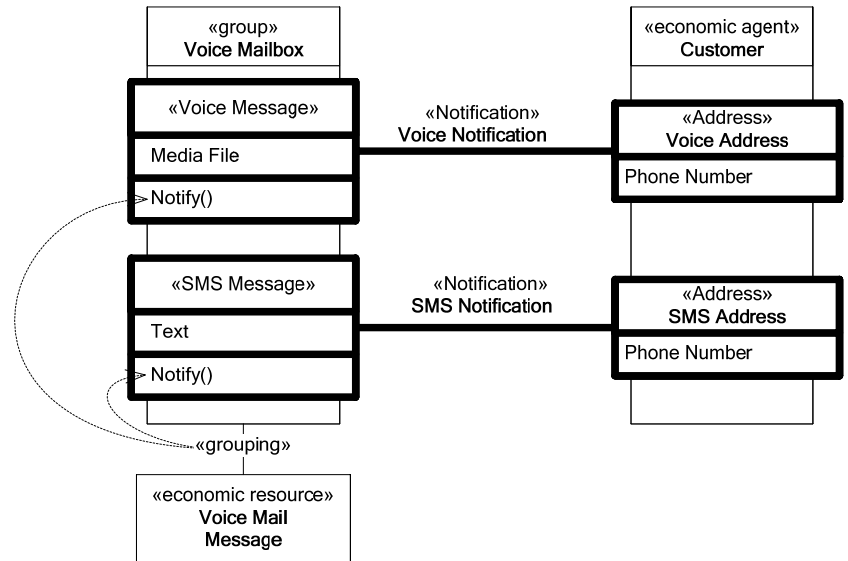

Fig. 44. Notification on new voice mail

Voice Mail Messages are economic resources that are members of the group Voice Mailbox of a specific customer. Whenever someone records a new voice mail message, the grouping relationship calls a Notify() method of the Voice Notification and SMS Notification elements. These elements create the voice and text messages and send them to the customer Phone Number.

6.5 INVENTOR'S PARADOX PATTERN

How do we discover a new service business pattern?

Context
REA is a modeling framework describing structure of business systems. The REA concepts have not significantly changed during last ten years; therefore, we do not expect any radical change in it in the near future.

In contrast, service patterns represent the functionality of the business applications that originate in user requirements related to specific procedures or technologies. It is natural to expect that users of business applications will require richer, more

powerful, and generally better software applications in the future. Therefore, it is likely that any limited list of service patterns does not meet all future requirements the users of a business application could possibly have. When application designers implement business applications, they are forced to discover new patterns originating from unexpected user requirements.

Problem

How to extend a business application in a consistent manner?

Forces

A solution is influenced by the following forces:

- Users of your business application require functionality that is not covered by the service patterns we know about.

- Users of business applications sometimes require very specific features that are not always good candidates for service patterns. Service patterns are generalized and reusable units of business logic; therefore, it usually requires substantial work to transform a specific user requirement into a business pattern.

- We would like a general rule or guidelines to help us formulate new business patterns from new user requirements.

Solution

The solution is known as Inventor's Paradox, described by the mathematician George Polya (Polya 1982):

> "A solution to a general problem is often simpler than a solution to a specific problem."[14]

In summary, the Inventor's Paradox is as follows:

- Solve a specific problem by solving a more general problem.

- The general problem paradoxically has a simpler solution.

- But you have to invent an appropriate general problem which covers your specific problem.

To apply the Inventor's Paradox, application designers analyze the users' business problems and try to extract patterns that can be generalized. Then, they solve this generalized problem as one or more service patterns. Finally, they solve each specific problem by configuring the service patterns in a software business application.

The guidelines above are general, and can be applied to solving problems in any domain. In model-driven design for software in a specific domain, the application developers must keep in mind the purpose of the domain, and generalize the

---

[14] Polya's original formulation was "The more ambitious plan may have more chances of success, provided it is not based on a mere pretension, but on some vision of the things beyond those immediately present." We use the formulation by Karl J. Lieberherr (Lieberherr 1997).

specific problems in a way that is consistent with the domain. This sounds easy; but, based on our experience, it is not.

We formulated the following guidelines to help application designers focus on generalizing specific problems in the scope of the business logic domain.

The service patterns in the REA modeling framework

- have business semantics,

- specify large units of functionality,

- their implementation often crosscuts the REA entities.

These principles are described in more detail below.

Service Patterns Have Business Semantics

"What business problem does this requirement solve?" is probably the most fundamental question to ask when examining a new user requirement. Users often tend to ask for a low-level or computational functionality, and it is up to the application designer to discover the real business purpose behind this requirement. For example, is a function that computes a sum of numerical values a good candidate for a service pattern in the business domain? Without domain-driven modeling in mind, a designer might think that he can generalize this requirement into an arithmetic operation pattern to cover subtraction, multiplication, and division as well. Would it be a good service pattern? We need to discover why the users need to sum values. Do the users need it for making an order total? Do the users need it for calculating the stock value of the product? The arithmetic operation is probably not a good candidate for a service pattern in the business domain, but contract total or account might be.

Is a currency converter a good candidate for a service pattern in the business domain? We need to discover why the users need a currency converter. If they need it for calculating the value of a payment in another currency, for calculating payment for international customers, and for calculating an offered price of the product, then monetary value will be a better candidate for a business pattern than a currency converter.

Service Patterns Specify Large Units of Functionality
If application designers develop a single business application for a specific purpose, they probably do not care about reuse. If user requirements change, the designers just change the application. However, if the application designers are developing a framework that will be used to configure several business applications in a product line, or to configure several very different business applications, then they would like to identify the functionality that is most complex and difficult to implement. Then, they can implement this functionality once in the reusable framework, and configure the actual software applications.

In such an environment, the more the complex and difficult functionality is implemented in the framework, the easier the job becomes for the application designers in configuring the actual business applications, and the less the overall amount of work (framework development plus application development).

Therefore, the more the larger, and most complex and most difficult units of functionality is implemented as service patterns, the easier the job of the application designers becomes. They can then focus on understanding and modeling users' business problems, rather than on implementing them.

Implementation of Service Patterns Often Crosscut REA Entities
Service patterns often crosscut structural patterns; therefore, if a user requires new functionality or a new data field on an REA entity, this will probably require some collaboration with data on other REA entities.

An example is address. In many business applications customer and vendor entities have addresses, such as shipping address and billing address. However, the addresses are also properties of the purchase order, sales order, and invoice. Therefore, it is useful to think of an address as a module having two elements: the default address on an economic agent, and the actual address on an economic event.

The address pattern presented in this paper even has different design, in which the default address is dynamically derived from historical information specified by economic events. Nevertheless, in both cases the address element crosscuts the entities that originate from the domain categories.

7 WHY DO WE NOT HAVE REA APPLICATIONS YET?

Actually, there are already some on the way.

REA Technology (http://reatechnology.com) has developed a model-driven REA solution that provides for full traceability of planned, expected and completed transactions; it is easy to understand by non-accountants as it uses the language the managers and other business decision makers are familiar with from their business. Due to the model-driven design, it can be tailored to meet specific needs of an organization an order-of-magnitude faster than any enterprise solution based on the traditional technology.

Workday (http://workday.com) is developing a new enterprise solution that can support the REA principles, according to Mark Nittler (http://www.aisvillage.com/rea25/mnittler.htm). Workday's product, also characterized as an on-demand alternative to ERP, provides features that current ERP solutions can't.

Group Accounting developed by Bob Haugen (http://group-accounting.com) is a new kind of business software for loosely-organized groups, believed to be one of the most important business forms of the future. Group accounting, similar to joint venture accounting, is difficult using traditional accounting, which can only be performed from the viewpoint of a proprietor. The REA model makes group accounting natural, as the REA accounting can be performed from the viewpoint of any single company in the group as well as an independent observer, simultaneously.

Although there are signs that the REA principles are becoming accepted, sometimes it takes a long time for new software technologies to become mainstream.

7.1 THE REA MODEL MIGHT BE A DISRUPTIVE INNOVATION

The term *disruptive innovation*, or *disruptive technology,* was introduced by Clayton M. Christensen and described in his books (Christensen 1997, 2003). It is a technological innovation, product or service that market leaders cannot deliver for **rational** reasons, either because their best customers do not want it, or because it would compete with existing and more profitable technologies. The technologies are called disruptive because they have to be developed by different companies than the established market leaders, and because they eventually overturn the existing dominant technologies in the market, the new providers might eventually displace the providers of the sustaining technologies. Examples of disruptive technologies are small-size hard disks, digital photography and the Linux operating system.

The REA model has many signs of disruptive innovations:

- Compared to prevailing technologies, it is simpler, smaller, more convenient to use, and cheaper in terms of effort required to develop specific applications

- It currently underperforms the solutions from leading vendors of enterprise resource planning systems in terms of available features and functionality, but this is rapidly changing due to other technologies that have recently become available.

- As leading firms' most profitable customers do not want and initially can't use products based on disruptive technologies, the current leading vendors of enterprise resource planning systems do not consider REA-based solutions a rational choice.

As disruptive technologies are usually filling a role in a new market that the older technology could not fill, business applications with embedded REA semantics have a great potential to open new market opportunities based on converged business models enabled by this new technology. The industrial impact will be achieved by universality of this technology and its ability to operate outside local markets.

The REA software applications will cause a major technological change in the way enterprises monitor, control and process their business information, with potentially huge economic impact. Compared to sustained technologies, where financial administration is based on 15th century principles, the REA model makes available an inexpensive model-driven platform with embedded generic business semantics, purposely designed to utilize the full potential of modern information technologies.

7.2 CSC CATALYST AND THE REA MODEL

We at CSC will benefit from knowledge of the REA modeling framework that can be used to design business solutions based on service-oriented architecture.
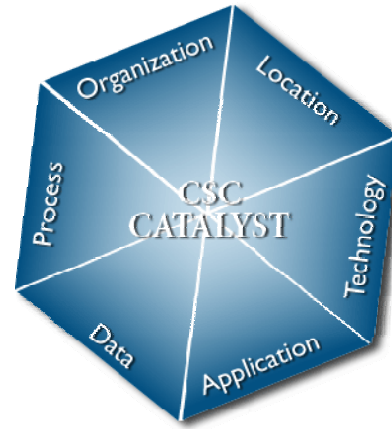

Fig. 45. CSC Catalyst domains of change

Fig. 45 illustrates CSC Catalyst domains of change. Business Area Architecture (BAA) asks questions in six domains that define the Problem Space (the requirements), and uses the Hexagon of Change as a means to ensure a complete assessment. These questions (BAA 2007) are outlined in the table on the next page, together with REA coverage of the answers to these questions.

Question	Area	Coverage by the REA Model
What processes are performed?	Process	*Covered.* The REA model gives precise answers that value-adding processes are performed, and includes consistency rules assuring completeness of the model. These rules might lead to a discovery of essential processes inadvertently not included in user requirements.
How are the processes performed?	Process	*Covered.* The REA model gives precise answers to how each process adds value to the company's resources. How this added value is technically achieved is specified by service models that extend the REA model.
Where are the processes performed?	Process	*Covered.* Location[15] is a service pattern that extends the REA model.

---

[15] Some researchers argue that location should be part of the core REA ontology, because events occur both in time and space. This approach called REAL (resources, events, agents, and locations) is described, for example, in (Hollander 1999).

Who performs the processes?	Organization	*Covered.* The REA model contains the concept of an economic agent, which gives precise answers to who is involved in the process, his or her role, and which economic events the agent is related to.
What services enable the solution?	Service	*Covered.* The REA model contains precisely defined extension points for services. The separation of the stable core and the changeable services enables design of long-living systems that can evolve over time.
What information do the processes require?	Data	*Covered.* The REA model is essentially a data model. Furthermore, the REA model gives precise answers to which data are essential for the customer's business (defined by ontological categories) and which data are supportive and might change over time (defined by services).
What will the user interface be like?	Application	*Covered.* The user interface can be automatically generated from the REA model, as discussed in the first chapter.
What technology supports the processes?	Technology	***Not Covered.*** The REA model is technology independent.

In addition to the BAA checklist, the REA model answers one other important question:

Why are processes performed?	Process	*Covered.* The concept of duality between economic events specifies cause-and-effect relationships between economic events. Therefore, the REA model leads to a system design that provides for full traceability of all transactions that influence values of economic resources.

This question cannot be answered by any other software design or modeling method.

It should be noted that the REA model provides precise but sometimes unspecific answers, as it has been illustrated in section 5, Capitalism for Software Engineers. Precision, rather than specificity, is essential in model-driven design. Furthermore, precise answers are very important in software design, as they prevent the modeler from creating incorrect models from a business perspective. In order to achieve specificity, the REA model must be extended by services, as illustrated in section 6, Extending the REA Model Using Services.

7.3 THE REA MODEL IN NATIONAL AND INTERNATIONAL STANDARDS

The REA model as a universal business ontology is very useful for specifying standards for electronic business. The REA model became the foundation for several electronic business standards, such as ebXML[16], UN/CEFACT[17] and Open-edi (ISO/IEC 15944-4)[18]. These standards have REA-based metamodels.

Although there are no direct legal requirements specifying a need for REA software, financial reporting requirements specified by the Sarbanes-Oxley Act can be easily met by REA software applications. Moreover, REA software provides for full traceability of <u>all</u> transactions that influence values of economic resources, not only financial transactions.

7.4 REA COMMUNITY

REA modeling is taught at many U.S. universities and business schools under the topic of Enterprise Information Systems. In Europe it is taught at the University of Gent, Belgium; Royal Institute of Technology in Stockholm, Sweden; and Technical University of Denmark.

American Accounting Association organizes yearly REA workshops called SMAP (Semantic Modeling of Accounting Phenomena), held every January since 2002. Half of the attendees is usually teaching-oriented and half is research-oriented.

The First International REA Technology Workshop was organized in Copenhagen, Denmark on April 22-24, 2004, and the Second International REA Technology Workshop was on June 25, 2006, on Santorini Island in Greece, in connection with the 3rd International Conference on Enterprise Systems and Accounting. The REA-25 conference was held June 13-15, 2007, in Newark, Delaware, USA (http://www.aisvillage.com/rea25/conference.html). An accounting interoperability workshop sponsored by the National Science Foundation and held in May 2008 in Washington had among the main topics REA, XBRL, accounting semantics, and formal specification of enterprise ontologies. ICESAL08 (http://icesal.org), held in July 2008 in Crete, Greece, had the REA model among its program topics.

A research-oriented mailing list is available at REATechnology@yahoogroups.com.

7.5 BOOKS ON THE REA MODEL

To date, four books on the REA model have been published.

Publication (Chang et al, 2007) shows how easy is to develop REA applications using Microsoft Access. Publication (Hruby et al, 2006) is targeted to software engineers and visionary managers at business software projects. Publications (Hollander et al. 1999) and (Dunn et al. 2004) are targeted to students of accounting.

---

[16] Electronic business eXtensible markup language, http://ebxml.org/

[17] United Nations Centre for Trade Facilitation, and Electronic Business, http://www.unece.org/cefact/

[18] ISO/IEC 15944-4:2006 Information technology - Business Operational View -- Part 4: Business transaction scenarios – Accounting and economic ontology.

ACKNOWLEDGEMENTS

REFERENCES

Appleton B (2000) Patterns and Software: Essential Concepts and Terminology, http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html

Arlow J, Neustadt I (2003) Enterprise Patterns and MDA: Building Better Software with Archetype Patterns and UML, Addison-Wesley Professional

Arnold TRJ (1991) Introduction to Materials Management, 3rd edition, Prentice-Hall Inc.

BAA (2007), BAA Essentials for SOA Final, PowerPoint slides for SOA Virtual Academy Course, CSC, 2007

Borch SE (2004) Typification in REA, First International REA Technology Workshop, Copenhagen 2004

Chang CJ, Ingraham LR (2007) Modeling and Designing Accounting Systems: Using Access to Build a Database, John Wiley & Sons Inc

Christensen CM (1997) The Innovator's Dilemma, Harvard Business School Press

Christensen CM (2003) The Innovator's Solution, Harvard Business School Press

Coad P, Lefebvre E, DeLuca J (1999) Java Modeling in Color with UML, Enterprise Components and Process, Prentice Hall PTR, New York

Cockburn A (2000) Writing Effective Use Cases, Addison-Wesley Professional

Coplien J (1996): Software Patterns, SIGS Publications, New York,

Czarnecki K. Eisenecker UW (2000): Generative Programming - Methods, Tools, and Applications, Addison-Wesley

David JS (1997) Three events that define an REA Methodology for Systems Analysis, Design and Implementation. Proceedings of the Annual Meeting of the American Accounting Association, 1997

Dunn CL, Cherrington OJ, Hollander AS (2004) Enterprise Information Systems: A Pattern Based Approach, McGraw-Hill/Irwin, New York

Evans E (2003) Domain-Driven Design: Tackling Complexity in the Heart of Software, Addison-Wesley Professional

Eriksson HE, Penker M (2000) Business Modeling with UML, John Wiley & Sons, Inc.

Fowler M (1996) Analysis Patterns: Reusable Object Models, Addison-Wesley Professional

Graham I (1998) Requirements Engineering and Rapid Development, Addison Wesley

Geerts GL, McCarthy WE (1997) Using Object Templates from the REA Accounting Model to Engineer Business Processes and Tasks. Paper presented at European Accounting Congress, Graz, Austria.

Geerts GL, McCarthy WE (2000a) The Ontological Foundations of REA Enterprise Information Systems. Paper presented at the Annual Meeting of the American Accounting Association, Philadelphia, PA.

Geerts GL, McCarthy WE (2000b) Augmented Intensional Reasoning in Knowledge-Based Accounting Systems. Journal of Information Systems, Volume 14, No. 2, 2000, pp. 127-150.

Geerts GL, McCarthy WE (2002) An Ontological Analysis of the Primitives of the Extended REA Enterprise Information Architecture" at http://www.msu.edu/user/mccarth4/

Greenfield J, Short K (2004) Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools, Wiley and Sons

Gruber TR (1996) A Translation Approach to Portable Ontologies. Knowledge Acquisition, 5(2):199-220

Gordijn J (2002) Value based requirements engineering: Exploring innovative e-commerce ideas, Vrije Universiteit Amsterdam.

Forman IR, Danforth SH (1998) Putting Metaclasses to Work, A New Dimension in Object-Oriented Programming, Addison-Wesley Longmann, Inc.

Haugen, B (2005) Resources and Rights, Discussion in REA Technology Mailing List, http://groups.yahoo.com/group/REATechnology

Hay DC (1996) Data Model Patterns, Conventions of Thought Dorset House Publishing, New York

Hay DC, Healy KA (2000) Business Rules, What Are They Really? The Business Rules Group

Hessellund A, Balthazar S, Chohan A (2005) REA-VMI Model, A General Framework for Vendor Managed Inventory (In Danish). MSc. Thesis, IT University Copenhagen

Henglein F et al (2006): Formal Specification of Commercial Contracts, Journal on Software Tools for Technology Transfer

Hollander AS, Denna E, Cherrington OJ (1999) Accounting Information Technology and Business Solutions, Irwin/McGraw-Hill

Hruby P, Kiehn J, Scheller CV (2006): Model-Driven Design Using Business Patterns, Springer

IDEF0, Integration Definition for Function Modeling (1993) National Institute of Standards and Technology, FIPS publication 183, http://www.idef.com/idef0.html

ISO/IEC 15944-4:2006 Information technology - Business Operational View -- Part 4: Business transaction scenarios – Accounting and economic ontology.

Jaquet M (2003) Realistic – A REA Model without Perspectives (In Danish). MSc. Thesis, IT University Copenhagen

Kiczales G et al (1996) Aspect-Oriented Programming, ECOOP 1996, Jyväskylä, Finland

Lampe JC (2002) Discussion of an Ontological Analysis of the Economic Primitives of the Extended-REA Enterprise Information Architecture. International Journal of Accounting Information Systems, March 2002 pp.17-34.

Lieberherr K J (1997) Inventor's Paradox, http://www.ccs.neu.edu/research/demeter/adaptive-patterns/AOP/IP.html

Marshall C (2000) Enterprise Modeling with UML: Designing Successful Software Through Business Patterns, Addison Wesley Longman, Inc.

McCarthy WE (1982) The REA Accounting Model: A Generalized Framework for Accounting Systems in a Shared Data Environment. The Accounting Review (July 1982) pp. 554-78

Mellor SJ, Balcer MJ (2002) Executable UML, A Foundation for Model-Driven Architecture, Addison-Wesley

Meyer B (1997) Object-Oriented Software Construction, second edition, Prentice Hall, Inc.

MDA Guide Version 1.0.1 (2003) OMG document omg/03-06-01.

Osterwalder A (2004). The Business Model Ontology - a proposition in a design science approach, Ecole des Hautes Etudes Commerciales, University of Lausanne, Lausanne.

Peyton-Jones S, Eber JM (2003): How to write a financial contract. In Jeremy Gibbons and Oege de Moor, editors, The Fun of Programming. Palgrave Macmillan

Polya G (1982) How to Solve It: A New Aspect of Mathematical Method, Princeton University Press

Porter M (1980) Competitive Strategy: Techniques for Analyzing Industries and Competitors, Free Press, New York

Rising L, Manns ML (2004) Fearless Change: Patterns for Introducing New Ideas, Addison-Wesley Professional

Rothbard MN (1978) For a New Liberty, Libertarian Manifesto, Collier Macmillan Publishers, London

Samuelson PA, Nordhaus WD (1989) 13 edition, Economics, McGraw-Hill, Inc.

Sowa JF (1999) Knowledge Representation: Logical, Philosophical, and Computational Foundations, Course Technology

Silverston L, Inmon WH, Graziano K (1997) Data Model Resource Book, A Library of Logical data Models and Data Warehouse Designs, John Wiley & Sons, New York, Chichester, Weinheim, Brisbane, Singapore, Toronto

UML 2.0 Superstructure Specification (2005), OMG document formal/05-07-04