FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Business Model Driven ERP Customization

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Wirtschaftsinformatik

eingereicht von

Thomas Gürth, BSc
Matrikelnummer 0725326

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Mag.rer.soc.oec. Dr.rer.soc.oec. Christian Huemer
Mitwirkung: Dipl.-Ing. Mag. Dr.techn. Dieter Mayrhofer

Wien, 13.01.2014

	(Unterschrift Verfasser)	(Unterschrift Betreuung)

Business Model Driven ERP Customization

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Business Informatics

by

Thomas Gürth, BSc
Registration Number 0725326

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.Prof. Mag.rer.soc.oec. Dr.rer.soc.oec. Christian Huemer
Assistance: Dipl.-Ing. Mag. Dr.techn. Dieter Mayrhofer

Vienna, 13.01.2014

(Signature of Author)

(Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Thomas Gürth, BSc
Hubertusgasse 14, 2410 Hainburg

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum)	(Unterschrift Verfasser)

Acknowledgements

Finishing this thesis would not have been possible without the help of many other people. These advisors and friends supported me with their opinions and inputs whenever I got stuck in a problem or seeked their counsel.

My advisors Christian Huemer and Dieter Mayrhofer provided me with their help and guidance during countless meetings and discussions. I would never have been able to finish a thesis that meets my personal expectations without their dedicated supervision.

I am also grateful for the support of my family and friends. My parents Manfred and Anita always provided me with valuable inputs and reminded me to relax when I spent too much time working. My special thanks go to Johanna for the discussions about my proposal and her suggestions on how to explain my thoughts to an audience that is not familiar with computer science.

In particular, I would like to thank my girlfriend Christina for her help and understanding whenever I was working late hours to achieve my goals. Without her dedication and love, writing this thesis would not have been possible.

Abstract

Enterprise Resource Planning (ERP) Systems encompass the administration of all information about resources that are needed for companies to run their business. They support several functional areas, like accounting, manufacturing, and sales in form of modules that are integrated by a single database where all business relevant information is stored. In order to guarantee a flawless and productive use of the system, the economic phenomena underlying the business of the company have to be reflected in the user interface as well as the data structure itself. Usually ERP systems are purchased by customers from vendors in form of standard software. Such a software supports a predefined set of functionality and has to be further customized to the specific needs of an enterprise, which is not a trivial task and often leads to additional costs. Furthermore standard software only supports specific processes that are based on best-practice assumptions of the vendors. Therefore adjusting ERP systems to changed market demands is hard since they are missing a business semantic base. Changed business needs can often only be represented by drastic changes in the data structure or the code which can lead to inconsistencies.

The REAlist project uses a model-driven approach to overcome the aforementioned problems with existing ERP systems and enhance their adaptability. Business needs can be represented in an easy and unambiguous way as business models. These models define the features and look of the ERP system. REAlist uses the Resource-Event-Agent (REA) ontology as a business modeling language since it was initially proposed to support the implementation of IT-Systems and is related to data modeling. REA allows the specification of events that have happened or are happening in the near future, resources affected by the events, and agents participating in the events. Furthermore, policies and commitments can be defined which are both important concepts of ERP systems. The underlying database of the REAlist project (REA DB) is based on REA and holds the business models as well as the business data. Its main part is generic, meaning that changed business needs can be applied to the ERP system without changing the data structure. Instead of using the classic class-diagram-like representation of REA, a domain specific language (REA-DSL) is used to simplify the creation of REA business models.

The aim of this thesis is to undertake the first steps of the REAlist project and create a mapping from REA-DSL business models to the REA DB. Furthermore user interfaces are automatically generated based on the saved models during runtime to reduce the effort that is needed for the customization tasks in existing ERP systems.

Kurzfassung

Enterprise Resource Planning (ERP) Systeme umfassen die Verwaltung aller Informationen über Ressourcen, die von Unternehmen zur Durchführung der Geschäftstätigkeiten benötigt werden. Hierbei werden verschiedenste Geschäftsbereiche, wie Rechnungswesen, Produktion oder Verkauf in Form von sogenannten Modulen unterstützt, die in einer gemeinsamen Datenbank integriert werden, um auch abteilungsübergreifende Geschäftsprozesse unterstützen zu können. Um eine problemlose Nutzung von ERP Systemen sicherzustellen ist es von großer Bedeutung, dass diese Prozesse sowohl im User Interface, als auch in der zugrundeliegenden Datenstruktur abgebildet werden. Da ERP Systeme heutzutage im Normalfall als Standardlösung implementiert werden, die einen vordefinierten Funktionsumfang bietet, ist das nicht immer der Fall. Anbieter von Standard Software basieren ihre Lösungen auf Erfolgsrezepten, die sich in der Vergangenheit bewährt haben und vor dem Einsatz an individuelle, unternehmensspezifische Anforderungen angepasst werden müssen. Derartige Anpassungen, die auch unter dem Begriff Customizing bekannt sind, sind nicht trivial und können hohe Folgekosten nach sich ziehen. Abgesehen davon kann Software, die im Moment funktioniert, oft nur sehr schwer an sich ändernde Marktanforderungen angepasst werden.

Das REAlist Projekt versucht die zuvor genannten Nachteile einzudämmen und die Anpassungsfähigkeit von ERP Systemen zu steigern. In diesem Zusammenhang wird ein modellgetriebener Ansatz verfolgt, der es erlaubt Geschäftsanforderungen als Geschäftsmodelle darzustellen, und diese zur automatischen Erzeugung der Benutzermasken so wie der Funktionen heranzuziehen. Zur Beschreibung der Geschäftsmodelle kommt die Resource-Event-Agent (REA) Ontologie zum Einsatz, die die Entwicklung von Softwareanwendungen unterstützen kann und ihre Wurzeln im Feld der Datenmodellierung hat. REA erlaubt die Definition von Events die bereits stattgefunden haben oder zurzeit im Gange sind, Ressourcen die von diesen Events beeinflusst werden, und Agenten die an den Events beteiligt sind. Weiters können Policies und Commitments beschrieben werden, welche wichtige Konzepte von ERP Systemen darstellen. Die Datenbank des REAlist Projektes (REA DB) basiert auf REA. Dadurch ist es möglich REA Geschäftsmodelle (und damit jede Form von Geschäftsanforderungen sowie Geschäftsaktivitäten) in der Datenbank abzuspeichern ohne deren Struktur abändern zu müssen. Um eine möglichst einfache Erstellung der REA Modelle sicherzustellen, wird eine grafische domänenspezifische Sprache (REA-DSL) verwendet.

Im Zuge dieser Arbeit soll ein Mapping der REA-DSL Geschäftsmodelle auf die REA DB definiert werden, das es ermöglicht die Geschäftslogik in der Datenbank abzulegen. Weiters soll, basierend auf den gespeicherten Modellen, eine automatische Generierung von Eingabemasken erfolgen, um den großen Aufwand beim Customizing zu verringern.

Contents

CHAPTER 1

Introduction

1.1 Motivation

Enterprise resource planning (ERP) encompasses the administration of all information about resources that is necessary for a company in order to execute its business processes. Examples for such resources may be material, employees, and finance. An ERP system covers multiple business sectors since the administration of several resources should be integrated by it as described in [24, 25]. Among several characteristics of ERP systems *dataintegration* might be the most important. This means that all applications use a common database [66]. Therefore modules like the ones offered by SAP (e.g., *human resources*, *production planning*, *sales and distribution*) are not isolated from each other. They are integrated by a single database where all applications have up-to-date information after an update occurred [54]. Current ERP systems are often purchased as standard software that supports a predefined functionality based on best practice assumptions of the vendors. For each customer an instance with a separate database, tailored to the specific systems, exists [34].

After the installation of a business standard software it cannot be used immediately. It first has to be adapted to the specific needs of the company. Two techniques are distinguished by Weber [66]: *configuration* and *development*. *Configuration* encompasses the traditional customizing (parameterization) where company specific configurations are done by setting parameters in an user interface and no programming skills are needed. Unfortunately a huge amount of customization possibilities exists and customizing cannot be done by everyone since it requires knowledge of the standard software itself as well as knowledge of the business processes of the company using the software. The customization of the systems has to be done by each tenant on its own. If the configuration possibilities of customizing are not enough, the functionality of standard software can be adapted by the *development* of own extensions.

Existing ERP systems are missing a business semantic base which would allow to define business needs directly in the system. This means that applications only support selected business processes of the industries and adjusting them to changing market demands is difficult.

Due to the aforementioned reasons companies struggle when trying to handle the complexity and flexibility of the day-to-day operations since the need to adapt to changed conditions constantly grows. Therefore the need for ERP systems with easier customization functions and the possibility of adapting to changed business needs in an easy way becomes evident. In this thesis we represent the business needs by business models that specify the economic phenomena on which companies base their business. These business models customize the features and user interface of the ERP system. Consequently, this thesis is titled `Business Model Driven ERP Customization`.

1.2 Problem Statement

ERP systems should be customizable in an easier way. The reason for this is that current customizing can be very complex and since a lot of the configurations cannot be changed once they have been done, high follow-up costs can occur. If the possibilities offered by the customization are not sufficient, own extensions of the system can be implemented. The drawback of this approach is that programming skills are needed and the overall complexity of the system is increased [66].

A system capable of adapting to changing requirements and needs in a more flexible way is also desirable. Current ERP systems are not adaptive enough to support new business needs and adjust to changed market demands. Often a serious change of the data structure or the code has to take place. Such drastic changes can lead to inconsistencies and hinder the traceability of data.

The REAlist project intends to overcome this drawback as proposed in [34]. Instead of providing separate instances of the ERP system to each customer, one single instance of the software, with one underlying database (REA DB), is hosted in the cloud. All customers access this single instance of the ERP system and consequently the same database which emphasizes the need for multitenancy (cf. Figure 1.1).

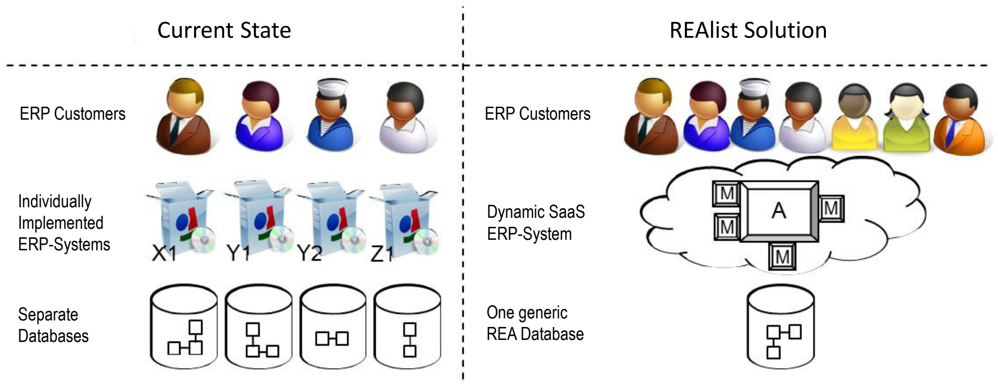


Figure 1.1: Current State of ERP Systems and intended REAlist Solution [34]

The main data structure of the database is generic. Therefore even if new business requirements are recognized and persisted, it does not need to be changed. As basis for the data structure the Resource-Event-Agent (REA) ontology is used. Introduced by McCarthy in 1982, REA is a generalized accounting framework that was developed to conceptualize economic phenomena [49].

Company specific business needs are described by graphical business models, which adapt the user interface and functions of the ERP system during runtime. These models are described by each company (tenant) of the ERP system using the REA-DSL. REA-DSL is a domain specific language also based on the REA ontology that enables the user to create business models in a clear and simple way [41]. The REA ontology includes important concepts of ERP systems and since the data structure is based on REA, each REA business model can be stored in the database. The feasibility of that approach has been shown in [44].

Existing business needs and requirements that are represented as REA-DSL models can be changed in a simple way. Since these models adapt the user interfaces and functionality of the system, the ERP's overall flexibility is increased. Furthermore the needed effort for customization is reduced.

1.3 Aim of the Work

Goal of the REAlist project is a cost-saving approach for company-specific adaptations or customizations of a multitenant ERP system. In order to enable simple company-specific customization a model driven approach is used. Since model engineering shifts the focus of software development from coding to modeling as described in [38], the REA-DSL business models become the main artifacts. Editing these models enables the user to quickly re-engineer its business and save them to the REA based database. Furthermore the saved models act as foundation for individual user interfaces that are automatically created. These user interfaces are used to record actual business cases that are also saved in the generic part of the database. Not all of the intended steps in [34] can be realized at once. Therefore only the parts described in Figure 1.2 will be implemented in the scope of this thesis.

1. **Creating database mappings based on the REA-DSL business models**: As described in [41] while developing the REA-DSL Dieter Mayrhofer mapped the models to a relational model in order to create the necessary tables. Thus, the business model was represented by the database schema. SQL files for the table creation were built using Microsoft Visual Studios T4 text templates [7]. In contrast, this work builds upon an existing REA database (REA DB) which can store the business models as well as the actual business data. We therefore do not create a new database schema but provide SQL inserts for the REA-DSL models that can be used to save them to the database. Since the REA DB and the business models are both based on the REA ontology, storing the models to the database is possible. During the development of the prototype the database that was proposed in [44] might be revised several times.

2. **Generating user interfaces based on the business models**: The creation of user interfaces that are based on the designed REA-DSL business models will be realized with the Google Web Toolkit (GWT)[1]. The emphasis of this task is not to generate extensive

[1]http://www.gwtproject.org/

and beautiful user interfaces. Instead it should be demonstrated, that the underlying REA database can be used to create an UI adapted to specific business needs. This user interface has to reflect all relevant concepts that are specified in the saved REA business models and enable users to conduct business cases. However, usability issues are not of primary interest. The entered business case data is stored in the generic part of the REA DB.

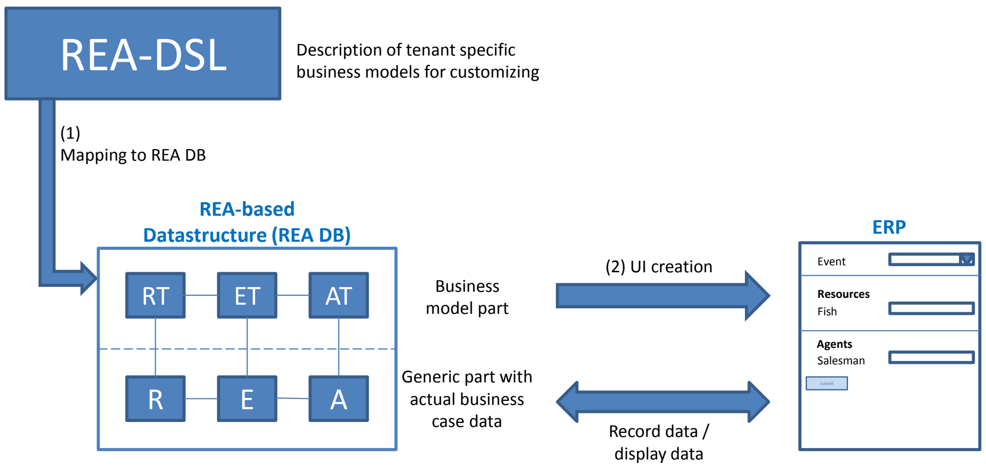

REA-DSL
Description of tenant specific business models for customizing
(1) Mapping to REA DB
REA-based Datastructure (REA DB)
RT
ET
AT
R
E
A
Business model part
Generic part with actual business case data
(2) UI creation
Record data / display data
ERP
Event
Resources
Fish
Agents
Salesman

Figure 1.2: Thesis Contributions

The approaches practicability in terms of how the business models are saved in the REA DB, as well as the completeness of the generated user interfaces should be evaluated. The main research question that should be answered in the context of this thesis is: “**how can REA-DSL business models be mapped to the REA DB?** “

1.4 Methodological Approach

The methodological approach that will be used in this thesis is *design science*. As Hevner et al. [31] describe, knowledge and understanding of the problem domain and its solution are achieved by the creation of an artifact. As suggested by Hevner, the two design processes *build* and *evaluate* will be conducted.

1. **Build process**

   a) **Literature survey.** A theoretical research of the important concepts used in this thesis, like the REA ontology, ERP systems, customizing, and the REA-DSL, will be done. Based on this research existing solutions regarding the usage of REA as basis for information systems and especially ERP systems will be studied. This should help to get a better understanding of the problem and build a foundation for the design of the artifact.

b) **Implementation.** The artifact itself will be implemented as a prototype that should be capable of the tasks defined in Chapter 1.3. To be more specific, based on predefined business models, mappings to the REA DB are created and an automatic creation of an user interface is performed.

2. **Evaluation process**

a) **Evaluation of the artifact.** The artifact is evaluated by testing the created database inserts from the mappings and the user interfaces based on twelve different business models that are used by McCarthy in his classes to demonstrate the various aspects of the REA ontology. These models are based on real companies in the United States and were already used by Mayrhofer [41] to demonstrate the functionality of the REA-DSL. Furthermore, we evaluate if the generated user interfaces represent all the concepts that are defined in the underlying business models and if they can be used to record actual business cases. Based on these outcomes possible extensions to the prototype and improvements, that could be performed in future work, will be pointed out.

1.5 Structure of the Thesis

The thesis is structured as follows. Chapters 2 to 5 describe the state of the art. In Chapter 6 to 8, the created artifact is explained and evaluated. Chapter 9 summarizes the work and points out further directions.

Chapter 2 describes what ERP systems are. After a general classification of these software types, their most important characteristics are examined, including their outstanding benefits, the ERP market, and solution strategies to realize ERP systems. Since most ERP software is purchased as standard software, this term is also described in greater detail. Furthermore, adaptation and customization of such standard software is extensively discussed. Finally, the significance of adaptability in the context of ERP systems is pointed out.

Chapter 3 explains the business model concept and the three most popular business models e^{3}value, BMO, and REA. Since the ontology that is used in the REAlist project is REA, it is discussed in greater detail. Therefore the basic concepts of resources, events, and agents are introduced. Built on this foundation, the concepts of value chains, commitments, types, and policies are elaborated.

Chapter 4 gives an overview of the REA-DSL. In order to demonstrate its usage an example company of a car producer is used. Based on the companies value adding activities, all the needed steps to create meaningful REA-DSL models are illustrated. Starting with the definition of the resources, agents, and their properties, the value chain is created. The included activities are described in greater detail by the planning- and operational view. At the end of this chapter readers should understand, what the important constructs of the REA-DSL are, why the REA-DSL makes the modeling of REA business models much easier, and why it is used in the REAlist project.

Chapter 5 presents the scope of the REAlist project. Since the REA based database (REA DB) constitutes the cornerstone of the project, the major part of this chapter deals with this topic.

First the generic part of the REA DB is illustrated, followed by a detailed description of the part that is capable of storing REA-DSL business models.

Chapter 6 deals with the storage of REA-DSL models in the REA DB. The mapping rules are illustrated graphically to explain the intended mapping in a simple way. Furthermore, algorithms are listed that describe the steps in greater detail. Based on the defined mappings, the creation of SQL inserts using T4 text templates is discussed.

Chapter 7 illustrates an approach to generate user interfaces based on persisted business model data. The GWT framework is used to implement the prototype which reflects the definitions of the underlying REA-DSL model and can be used to save business case data in the REA DB.

Chapter 8 evaluates the created prototype. Since the artifact should be capable of mapping REA-DSL business models to the REA DB and automatically generating user interfaces, both conditions are tested. The created SQL inserts have to cover all concepts that are defined in the business models. Furthermore, the statements have to execute flawlessly and populate the REA DB without errors. On the other hand the generated user interfaces have to comprise all elements that are needed to conduct business cases and give users the opportunity to save, update, or delete data in the REA DB. To accomplish this, indicators are defined that facilitate the decision if the demanded requirements for the generated SQL scripts and the user interfaces are met. The UI-Generation prototype will be evaluated using the same approach.

Chapter 9 summarizes the main parts of this thesis. Furthermore, limitations of the created artifact and further directions are discussed.

CHAPTER 2

ERP Systems

2.1 Terminology

ERP systems belong to the family of enterprise application standard software (or business standard software) and can further be categorized as integrated function software. Meister [51] gives an overview of different software types (cf. Figure 2.1) where she defines application systems as complex software that enables users to work on specific tasks like processing an order, reading emails, or recording data relevant for business cases. System software controls the computer hardware and therefore builds the foundation for running the application software. Application software is tailored to a specific domain. Enterprise application systems (EAS) therefore support users in business related tasks. According to Fowler [12], they display, store, and manipulate large amounts of data to support business processes. Such processes are partially ordered sets of tasks or steps that are undertaken towards a specific goal [9]. According to Davenport [10], they are a structured, measured set of activities designed to produce a specific output for a particular customer or market. Examples for other application software are computer games, image processing tools, or learning-programs. Application software for enterprises can either be developed individually for the company and its specific requirements, or using off-the-shelf solutions by vendors like SAP, that support a well defined application area and have to be adapted by the customers to their detailed needs. Such a standard software can either be tailored to commercial and administrative duties, or to other areas like CAD-modeling (Auto CAD) and software development (eclipse, visual studio .net). In the first case standard software can be divided into industry-, special-, and function software. While industry-software encompasses solutions for specific industries like health care, manufacturing, or banking, examples for special-software are office applications like Word, Excel, or PowerPoint. Function software either is developed for single functions (accounting, human resources, purchasing, sales, production, and logistics) or integrates business relevant tasks into ERP systems. These systems can support such tasks in several business sectors and are investigated further in this thesis.

The acronym ERP stands for enterprise resource planning and can therefore be interpreted as the planning and organization of all resources affecting a companies business. The first arising

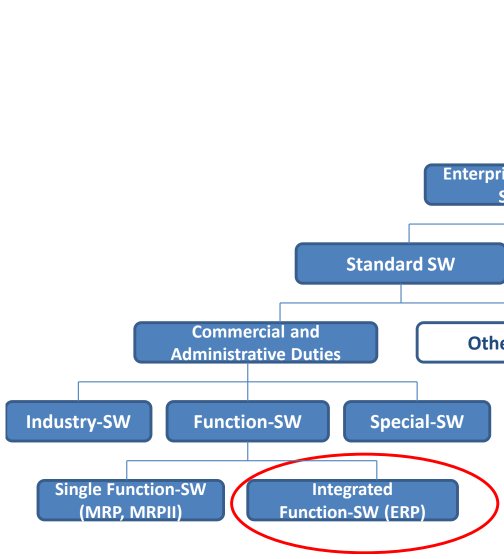

Software (SW)
Application SW
System SW
Enterprise Application Systems
Other Application SW
Standard SW
Individual SW
Commercial and Administrative Duties
Other Duties
Industry-SW
Function-SW
Special-SW
Single Function-SW (MRP, MRPII)
Integrated Function-SW (ERP)

Figure 2.1: Classification of Softwaresystems [51]

question that needs to be answered in this context, is what such resources are. According to Gronau [24] an ERP system encompasses the administration of information about raw material, employees, production capacities (like used machines), or finances. He argues that, compared to specific application systems like the aforementioned single function software tailored to areas like production or accounting, an ERP system should administrate at least three of the named resources. According to that definition resources are all objects that are needed to handle an enterprises day-to-day business processes. ERP systems manage this mass of data and support all functions and processes of a company [2]. Not only one functional area is supported. ERP systems can encompass all, but at least the most important ones like finance, human resources, logistics, and production [30]. The components of the ERP supporting these areas are called modules. According to Grammer [23], four main modules exist: *purchase & logistics*, *human resources*, *finances*, and *sales*. SAP, the ERP vendor with the largest market share, offers much more like *financial management*, *controlling*, *treasury*, *project system*, *production planning*, *plant maintenance*, *sales and distribution*, *human resources*, and *materials management* [54]. Due to technological advancements like the internet, ERP systems can also be used inter-organizational between enterprises.

Before the advent of ERP systems, software supporting business processes was implemented as individual solution for specific companies. In the beginning such systems were mainly developed for the accounting domain. In order to plan and manage manufacturing processes the first standard software solutions were introduced in the late 1960s in form of MRP-Systems (*Material Requirement Planning*). These systems enable managers to plan what resources are needed to produce products according to bills of material at the right time [24, 39]. They support companies to keep their inventory levels as low as possible while ensuring that enough material is in stock to produce products for customers. Due to advanced hardware technology more functions were integrated into MRP-Systems, which lead to the new term *Manufacturing Resource Planning*

(MRPII) [36]. Compared to MRP, that only focuses on material needed for manufacturing, MRPII-Systems encompass the whole production line. They include functions like purchase, capacity planning, and production planning and control [24]. As already explained ERP systems integrate even more aspects that are not directly related to production but are necessary for a company to do business.

2.2 Characteristics of ERP Systems

Typical characteristics of ERP systems are function-, process-, and data integration, intra-enterprise processing and the focus on the daily business. The minimal scope of integration that needs to be supported is data integration, which means that one common database is used by the overall system to support cross-functional business processes and avoid data redundancy [24, 54]. The application domain of ERP systems encompasses, among others, purchasing, production planning, inventory management, material requirements planning, order inflow, invoicing, sales analysis, payables, receivables, general ledger, budgeting, financial consolidation, payroll, benefits, and recruiting [24]. This listing of applications also emphasizes that ERP systems cover multiple functional areas. Logistics is supported as well as human resources and accounting. Figure 2.2 shows another view of ERP application domains based on Grammers [23] interpretations. The complete functionality of an ERP system can be summarized under four pillars: purchase & logistics, human resources, finance, and sales. Data integration is achieved by using one common database that stores all business relevant information and is used by all modules.

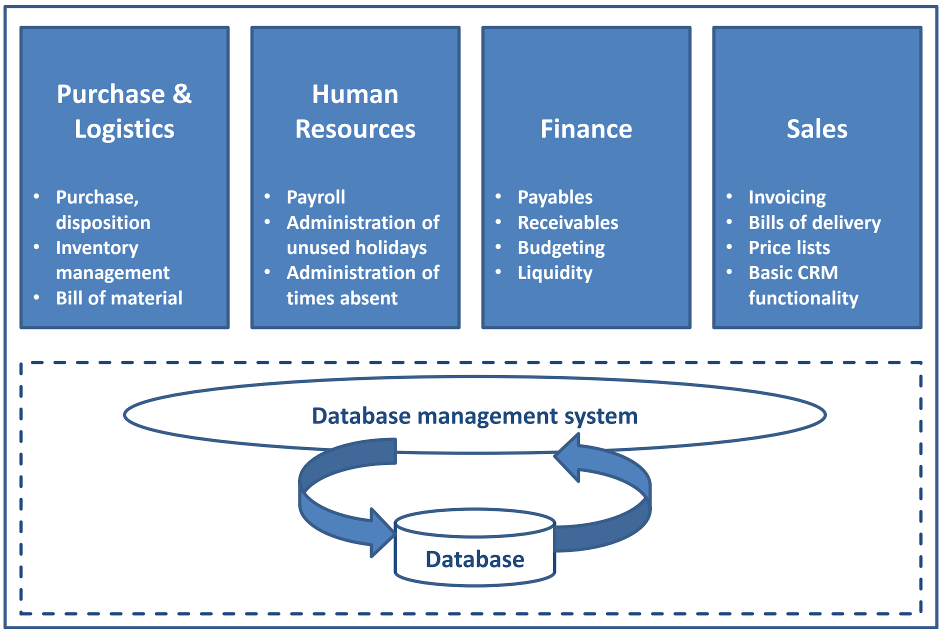


Figure 2.2: Application Domain of ERP Systems [23]

Integration

As already indicated, data-, function-, and process integration are essential parts of ERP systems, that clearly distinguish them from isolated applications for a specific domain like the aforementioned single function software for accounting, purchasing, or production.

Data integration was already explained and is defined as storing all business data into one database that will be used by the integrated system. When enterprises use a variety of different applications with their own data repositories, data has to be saved several times. Even slight errors can therefore lead to inconsistent data. Exchanging and synchronizing the data among the different systems can lead to duplicates and conflicts. Furthermore, reacting to changed business data becomes difficult. Data integration builds the foundation of every system integration [51].

Function integration combines several functional areas. Examples are the management, human resources, finance, distribution, purchase, production, logistics, and sales. While logistics, purchase, production, sales, and distribution are parts of the basic business processes influencing the day-to-day operations of an enterprise, human resources and finance handle administrative tasks. The management defines rules and controls all other functional areas. Therefore three different layers of functional areas can be identified: operational, administrative, and strategic (cf. Figure 2.3). In order to integrate functions, an enterprise application system combines functional areas from these layers. *Horizontal integration* happens on the same layer. If several functions on different layers are combined we talk about *vertical integration*. ERP systems are at least horizontally integrated. During the integration, functions are coordinated to enable that results of preceding ones can be further processed by the succeeding functions [24]. To accomplish this, several functional areas have to be based on the same database, which requires an already existing data integration [51].

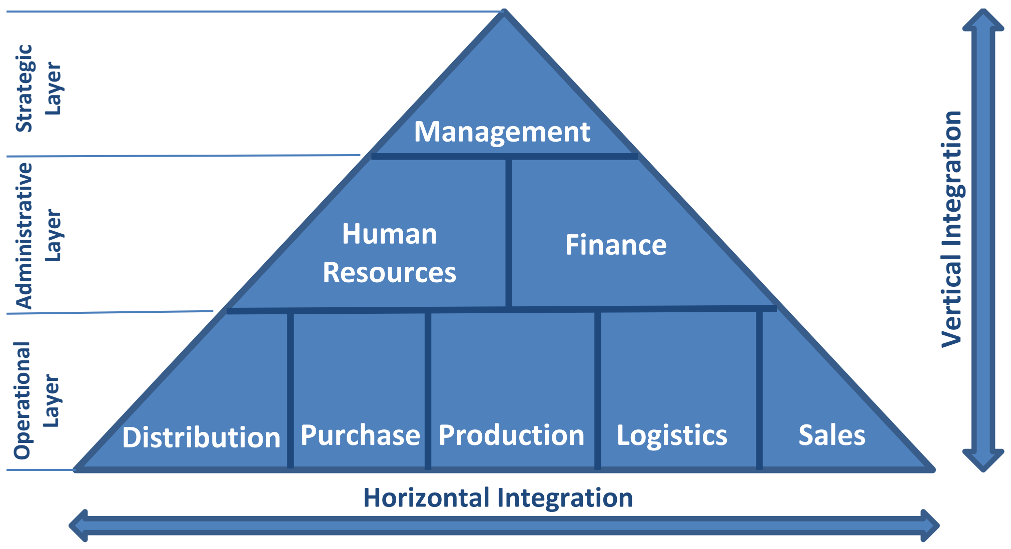


Figure 2.3: Function Integration in Application Software [51]

While function integration just requires that several functional areas are combined in one system based on the same data, **program integration** is achieved when the modules of the ERP system representing these areas are able to communicate with each other. Workflows define when a request from one module to the other happens due to specific events. An example is the credit check of a customer. If a customer buys an unusual large amount of finished goods the sales module can request a check of the customers creditworthiness from the finance module [51].

Process integration couples business processes to an automated system-activity. This is achieved by using standardized best-practice processes and cross-functional transaction automation. Process integration is the highest form of integration that can be achieved in an ERP system [51].

Mertens [52] classifies data-, function-, and process integration as *integration subjects* and proposes *integration range*, *integration direction*, and the *scope of integration* as additional differentiation criteria.

The range determines if integration happens intra- or inter organizational, only in specific functional areas, or between areas. Business processes can include information from several functional areas. For each of these areas separate application systems can be used (e.g., single function software for purchase, human resources, production, and logistics). The more application systems are integrated based on the common database, the higher is the range of integration. Inter-organizational integration can be achieved through extensions like *customer relationship management* (CRM), *e-procurement*, and *supply chain management* (SCM), that enable a usage of the ERP between companies and customers. ERP systems that offer such interfaces on the buy- and sell side are termed ERPII [24]. Figure 2.4 illustrates this fact. While SCM facilitates better operational and business planning by coordinating the flow of material and information among supply chain partners, CRM provides an infrastructure to establish long-term relationships with customers [29]. E-Procurement systems can be used on the buy side to purchase products that are needed by the enterprise to perform its business. The emphasis of the actual ERP lies on the internal business processes of the company. Systems with CRM, SCM, and e-procurement extensions therefore have a higher integration range.

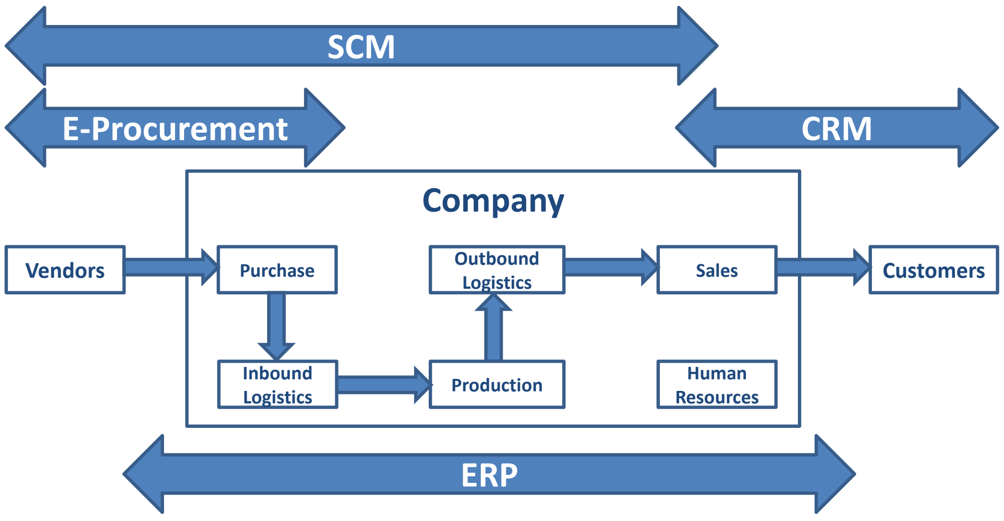


Figure 2.4: Extensions of ERP Systems [24] (adapted)

The direction of integration was already identified before in the scope of function integration as horizontal and vertical. It determines if a system integrates functions only on one hierarchy layer, or several ones (cf. Figure 2.3).

An ERP can either have a complete or a partial integration scope. Partially integrated systems focus on specific functional areas or combinations thereof. Examples are *finance and accounting*, or *production and logistics*. A completely integrated ERP system encompasses all functional areas.

These specifications make clear that companies planning to implement an ERP have to define what should be supported by it to what extent. Enterprises that focus their daily business only on selected functional areas will not need to use all available modules. Extensions like SCM or CRM will also not be needed by every customer. It therefore is important to clearly define what the ERP system should integrate in order to support the individual processes of the company and improve its business performance.

Benefits of ERP systems

When replacing legacy systems or single function software by enterprise application systems like ERP systems, enterprises can benefit from it in several ways. Hendricks et al. [29] argue that the integration of information is a key benefit of EAS since the replacement of poorly connected legacy systems that only support specific functional areas reduces support costs for the infrastructure. They furthermore state that due to the operational integration of the functional areas the performance of the entire company can be positively affected. The replacement of complex interfaces between legacy systems with standardized automated cross-functional transactions reduces *order cycle times* and therefore increases the *throughput*, *customer response time* and *delivery speed* [8, 46]. Furthermore, automating financial transactions reduces *cash-to-cash cycle times*. Since data integration is a requirement in ERP systems all relevant data is only stored once centrally in the database. Since all modules are based on this database they have up-to-date information when an update occurs. This also affects the planning since plans can be created that reflect current conditions of the business processes. Ensuring that operational processes remain synchronized enables a company to provide customers with updated information about their orders [3]. The standardized automated cross-functional transactions and the central database, encompassing all relevant enterprise data, enables users to observe the performance of the company and its parts. This gives managers the opportunity to identify potential improvements and take advantage of these opportunities [29]. Gronau names a variety of benefits when using ERP systems in [24]. His views are shown in Table 2.1.

One outstanding benefit of ERP systems is the standardization of business processes which can enhance productivity. The coordination of tasks can be simplified and the risk of mistakes/failures is reduced since the error source can be prevented in the first place by adopting best practices. On the other hand standardization increases the risk that users can not appropriately react to unexpected events since the possible actions in standard solutions can be too restricted. Gronau [24] also warns that employees will experience motivation- and identification problems when autonomous decisions are not possible. He argues that this could hinder the further development of the employees. Chapter 2.3 will explain standard software in greater detail.

	Without ERP usage	**With ERP usage**
Throughput time / lead time	Cost-intensive bottlenecks	Time reduction and lower costs due to optimized business processes
Order Processing	Processing needs data in several legacy systems with own databases (customers, products, orders)	Integrated database eases order processing and saves time. Updates only affect the common database
Financial Situation	Costs due to long positions or high receivables	Inventory control and dunning enhance operative performance
Business Processes	Multiple efforts in process execution	Optimized business processes based on best practices
Productivity	Responding to supplier- and customer needs takes a considerable amount of time	Improvements through adapted, optimized business processes, integration of CRM and liquidity management
Supply Chain Management	No integration with supply chain partners	Coordination of material- and information flow among supply chain partners
E-Business	Web interfaces as isolated systems	Integrated web-interfaces form the front-end of the ERP system. Usage of E-Procurement systems on the buy-side to purchase products from suppliers
Information	No efficient monitoring and control of enterprise resources	Data integration enables cross-functional access to the same database for planning and control
Communication	No efficient communication with suppliers and customers	Enhanced communication between the enterprise, suppliers, and customers due to extensions like SCM and CRM

Table 2.1: Benefits of ERP Usage [24]

ERP Market

A great number of ERP vendors exists. Gronau [24] therefore categorizes the existing market into the *range of functions*, the *degree of specialization*, the *number of system users*, and the *regional propagation*. The **function range** of an ERP system encompasses the resource administration of functional areas like finance, human resources, logistics, and production. Business processes occurring in these functional areas are optimized and supported by the system. ERP systems can either integrate a finance module for accounting purposes, or interconnect with external systems that fulfill the needed purpose. This extends to human resource modules or CRM functionality. While the **number of system users** describes if the software is used in single enterprises, small or medium sized enterprises, or large corporate groups, **regional propagation** depicts in which geographic regions the system is used. In this context especially ERP systems for corporate groups can be very complex since this groups unite several subordinate enterprises under the leadership of one company. Management seeks to steer the corporate group by financial indicators that express the productivity of each subordinate enterprise. Therefore such ERP systems encompass powerful solutions for finance, accounting, and controlling. Corporate groups often use a variety of ERP systems that need to be coordinated and harmonized in order to reduce the operating- and service costs. Compared to that, solutions for small companies only have a small subset of that functionality. But even small businessmen frequently need to organize their inventory invoices where an ERP system can be supportive. Finally the **specialization degree** describes if the software focuses on a specific industry sector like energy, meat processing, or plant manufacturing. Cross-industry solutions can be used in several industries.

The overall market for ERP systems is continually growing. According to [5] a growth of 2.2% was observed during the year 2012 and especially Software-as-a-Service (SaaS) applications shows potential for further growth. The five outstanding vendors in this sector are *SAP*, *Oracle*, *Sage*, *Infor*, and *Microsoft*. Together they hold 55% of the overall ERP market. SAP retained the worldwide market share leadership with 25%. Figure 2.5 visualizes these facts.

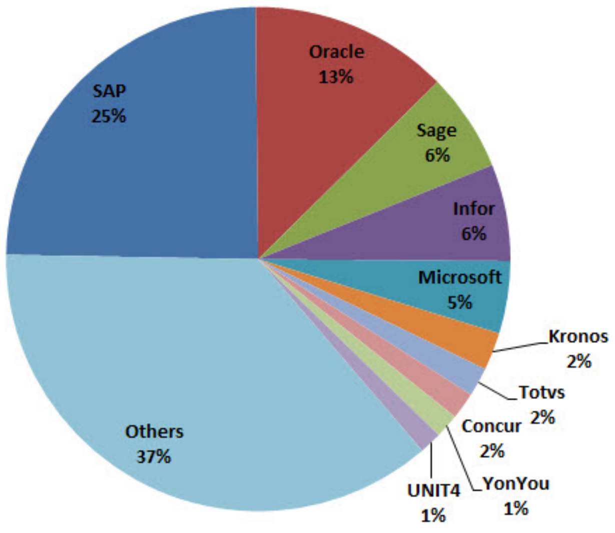


Figure 2.5: Market Share of ERP Vendors [5]

The dominant vendors in each of the four aforementioned categories differ a lot. Large vendors that are dominant in one category may be surpassed by smaller vendors in another category. Especially highly specialized software for smaller companies can offer market opportunities for smaller vendors. Since the big players do not intend leaving clients to competitors, they adapt their portfolios to increase their market share. An example is SAP, that focused on large companies and corporate groups with its ERP system *SAP R/2* and the successors *SAP R/3* and *SAP ERP 6.0*. Since the functionality of this solutions was far to large for smaller companies, the product range was extended for small and medium sized companies. *SAP Business All-in-One*, *SAP Business One*, and *SAP Business By Design* can be named in this context [23].

According to [5], especially the revenues of SaaS-based ERP solutions are going to grow in the upcoming years. Compared to on-premises software, that is installed on computers on the premises of users, the ERP system is hosted in the cloud. Reasons for an increasing adoption of that approach are that companies perceive its *total cost of ownership* (TCO) lower than of on-premises software. Furthermore, it is thought to be easier and faster to deploy [6]. SAP Business By Design is an example of an ERP system, that is offered as SaaS. It is an internet based on-demand-software for medium sized companies, where hardware and software are located in data centers anywhere in the world. Users access the software on demand via the internet [23].

Solution Strategies for Enterprise Application Systems/ERP systems

There is a variety of different ways to implement enterprise application systems, or more specifically ERP systems. Meister [51] distinguishes between *isolated solutions*, *proprietary client-server solutions*, *web-based client-server solutions*, *browser-based solutions*, and *webservices*. Following her definitions a more detailed description is given in the following. In **isolated solutions** the system is locally installed on a single workstation (or a small number of workstations) of the enterprise. An integration with other application systems does not exist. Therefore such a strategy will not be considered for modern ERP systems that integrate functions like SCM and CRM. Client-sever solutions distribute functions that are located on one and the same computer in isolated solutions (like user interface, data administration, and application logic) on several computers (server and clients). Multiple forms of distribution are possible. For instance only the graphical representation can be located on the clients, while the main part of the application is located on the server. Another possibility is to locate the representation as well as the application logic on the client and only locate the data management on the server side. In general, one can argue that clients access the servers functionality and data over user interfaces. This enables several users to access the same system and therefore manipulate the same data. Since this is essential for ERP systems, Gronau [24] states the client-sever solution as traditional architecture for them. **Proprietary client-server solutions** are completely installed in the internal network (LAN, intranet) of the enterprise using the system. Therefore the enterprises are responsible for tasks like data updates, backups, and data administration. **Web-based client-server solutions** on the other hand outsource the server to a provider that hosts it. Only the clients remain in the internal enterprise network. The provider is responsible for maintaining server operations, data administration, and performance. Enterprises are only responsible for the access rights of the clients and the maintenance of their internal network. **Browser-based solutions** are similar to that, but compared to the aforementioned strategy a simple web browser acts as client. The ERP system can be hosted in the cloud and users are able to access it via the internet. This has the advantage that no additional software has to be installed locally on the computer. Such an approach, where software like ERP systems is available as service in the internet, is called Software-as-a-Service [34]. The **webservice** strategy seeks to reuse standardized software routines that are offered as service in the internet. Meister [51] argues, that very detailed specification and modeling of the underlying business processes is mandatory for such a solution. Otherwise similarities in the processes that can be supported by offered services are hardly identifiable.

2.3 Standardsoftware

The similarities between the business processes of different enterprises enabled the success of business standard software in the first place since the reason for the development of such software are customer groups that share common needs. In the case of ERP systems these needs are the improvement of business processes and routines of the company. Furthermore, the process-oriented communication between functional areas should be enhanced [23]. An example for that is the value-added chain in the manufacturing industry where, regardless of the product type that is manufactured, from an economic point of view the same processes and interdependencies can be identified between purchasing, warehousing, production, finance, and sales. Figure 2.4 shows this value-added chain as internal processes of a company. If the essential business processes of several enterprises would not be quite common, the development of standard software never would have happened to that extent [23, 30].

The functionality of ERP systems can also be implemented as individual solution, but there are a lot of arguments that suggest the usage of standard software instead. Such a software is not developed for an individual customer and its requirements, but the needs of as much customers (in this case enterprises) as possible. Best practice processes are used as reference models, that build the foundation for the development. Since the standard software therefore is able to fulfill the needs of many enterprises, the development costs can be distributed between them, which enables vendors to offer their solutions for little license costs (compared to individual solutions) [30].

Besides that, the main advantage might be that standard solutions by prestigious vendors like SAP or Oracle are available very fast. Compared to individual solutions, enterprises just have to specify what functionality (modules) should be supported by their system of choice and purchase the product. Since vendors like SAP have already sold their product to a lot of customers, a flawless usage of the system should be guaranteed. Therefore, compared to the implementation of an individual solution, the threat of losing money is minimized. The range of functionality that is supported by the standard solutions of the largest ERP vendors is widespread. As mentioned in Chapter 2.1, SAP offers a great amount of modules that span all functional areas of an enterprise. Existing solutions are integrated to a large extent. Achieving such an integration level in individual solutions would take a huge amount of time and money. When standard software is implemented a huge amount of documentation is delivered. Prestigious vendors also hold additional trainings for the system usage and offer a hotline that can be called when problems occur. Furthermore, all parts of a standard system have a common terminology. Adaptations to such a system are also possible since vendors enable customers to extend the solutions to their own needs (cf. Chapter 2.4).

The decision to use a standard software also has disadvantages, but these are usually outweighed by the advantages and opportunities that exist. It is not a trivial task to describe the business processes of a specific enterprise in a standard software. Especially depicting extraordinary process aspects and integrating external systems is difficult. Since standard software is developed by external vendors, release-changes as well as as performance- or storage space problems can not be influenced and may hinder the functionality. When adapting standard software to specific needs enterprises have to depend on external consultants. Furthermore, standardized user interfaces can be hard to handle for the users which leads to problems of acceptance [24]. Some of these disadvantages can already be compensated by available adaptation methods for standard software that will be explained in the following.

2.4 Adjusting Standard Software to Enterprise Needs

An installed business standard software can not be used immediately. The reason therefore is that the software is based on best practices that can be adopted in nearly every business. Although standard software can have an extensive amount of functionalities, this does not mean that all the individual requirements of an enterprise are covered by it. McNurlin et al. [50] argue that the business processes that are supported represent the vendors assumption about the business. If these assumptions greatly differ from the views of the customers, an implementation of the ERP system will not be successful. The software first has to be tailored to the specific needs of the customers. Vendors of standard software therefore include possibilities that reduce the effort, needed to adapt the system, to a minimum [30]. This process is called *customizing*.

Customizing

Right after a standard software has been delivered to the customer, the first thing that has to be done is setting up the client. In this context the static (the organizational structure) and dynamic (the process operations) enterprise aspects are depicted in the standard software. Examples are company specifications (banking connections, shipping address), country-specific settings (currency, address code format), accounting parameters (tolerance limits, rounding methods), or enterprise specific objects (chart of accounts, business year).

Such system configurations are called *parameterization* since parts of the standard software are enabled or disabled by setting parameters in tables or other application objects and no programming skills are needed. A huge amount of different settings exists for all modules of a standard software like purchase, sales, production, or human resources. This leads to a large number of customization possibilities. Since customizing has to be done for every customer, vendors are interested in simplifying this task as much as possible and deliver checklists together with the standard software that describe all the tasks that have to be carried out in detail. These lists can be used as guidelines that help the customers to set up their purchased product and guarantee a productive and flawless usage. Unfortunately these measures do not reduce the scope of the customization tasks and especially adapting ERP systems with a wide functional range (integrating a large number of modules) can be very complex and take a considerable amount of time since a huge amount of tasks has to be carried out. Furthermore, setting up clients can not be done by everyone. Knowledge of the standard software as well as the business processes of the company is needed to set up clients. Therefore external consultants are often involved in the customization process, leading to additional costs for the company. Costs for trainings and release-changes also have to be taken into account.

Customizing in form of parameterization is the preferred way to adapt a standard software to individual needs since the standard can not be affected in an unintended way. This guarantees that the software will still work in the intended way after a release change by the vendor occurs. While customizing encompasses tasks that need to be done in order to use the system and does not require programming, there are other adaptation possibilities that are not necessary but can help to increase the coverage degree of standard software. Such techniques demand programming skills since own extensions are developed that fulfill highly specific company needs. Major ERP vendors therefore include an integrated development environment (IDE) in their products that enables customers to do so. Hesseler [30] distinguishes three different techniques: *proprietary developments*, *extensions*, and *modifications*. These methods are explained in the following.

Proprietary Developments and Extensions

Proprietary developments are implemented when the standard software is not capable of supporting the needs of an enterprise and the customizing options are also unsatisfying. Such solutions have the advantage that they are fully independent from the standard. This means that even when new versions of the standard software are released, the proprietary developments do not collide with it and can further be used. Proprietary developments should not happen too often since they are comparable to individual solutions and are expensive. When too much functionality is added in this way, implementing an individual solution in the first place might be the better choice.

Extensions are similar to proprietary developments but also influence the standard. Due to the high integration degree of ERP systems, adapted application objects can have unintended effects on functional areas they don't belong to. Therefore, integration tests have to be conducted when extensions are developed for standard software.

Modifications

Compared to the aforementioned adaptation techniques, modification is riskier and should be avoided if possible. While proprietary developments and extensions include safety mechanisms that ensure that no unintended changes of the standard are conducted, modifications can change the entire software functionality. Another criteria for not using modifications is the fact that such adaptations are lost when a release change happens. Extensions are normally still supported after such a change.

Each of these adaptation techniques increases the coverage degree and causes additional costs. Figure 2.6 visualizes the available techniques with respect to these two factors. Individual Software has a high coverage of the company needs but also is very expensive. Proprietary developments and extensions can significantly increase the coverage degree of standard solutions for relatively low additional costs. On the other hand risky, complicated modifications seldom increase the coverage degree a lot. Furthermore, the additional costs that can occur every time a release change happens can hinder the return-on-investment of standard software and make it even more expensive than individual software.

Personalizing

While customizing, proprietary developments, extensions, and modifications are functional adaptations on the enterprise level, the goal of *personalizing* is to grant an efficient system usage for individual users or user groups. The importance of this is obvious since software that is hard

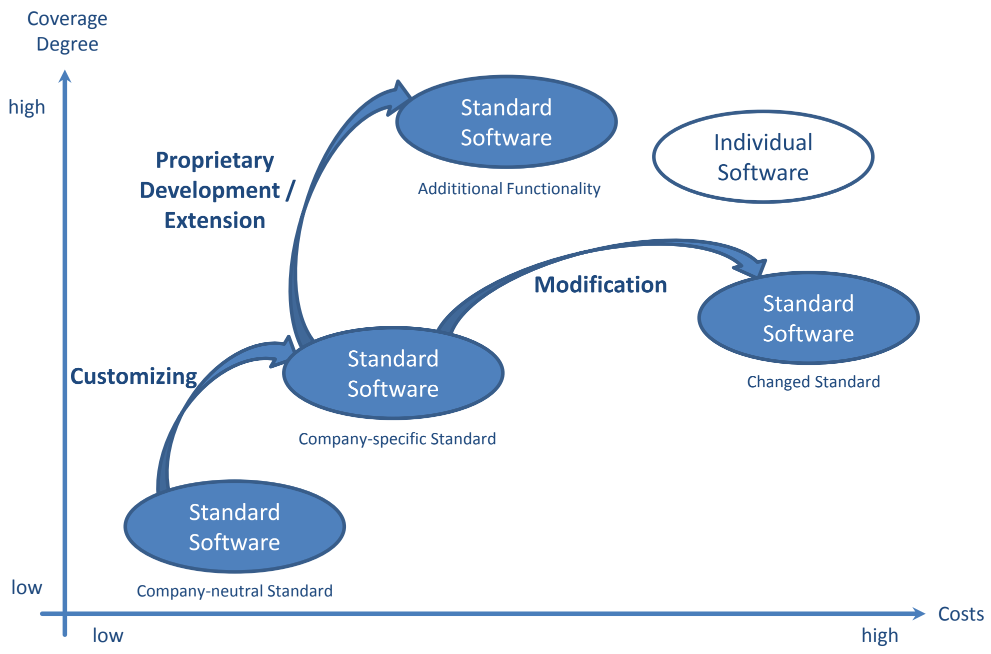


Figure 2.6: Customizing Techniques of ERP Systems [30]

to use or understand will hardly be accepted by the people working with it. Personalizing can be used to overcome this by achieving an individual look and feel of the user interface by shortcut creation for user relevant tasks or disabling screens that are not needed by certain users. The latter one even reduces the functionality of the software in exchange for a better usability.

Since personalizing is not necessary and its advantages can not directly be quantified due to the fact that it has no influence on the business functionality, companies often refrain from using this option to save costs, which leads to poorly adapted systems. Hesseler [30] argues, that *role based concepts* are a promising approach to further reduce the effort needed for personalizing. The concept is based on the definition of roles describing requirements, needed functionality, tasks, and authorizations of positions in an enterprise. Based on these roles vendors of standard software are able to adapt the software to specific users. If the role descriptions conform to the actual roles in the enterprise, personalized user interfaces of individual end users only encompass the functionality that is needed to conduct their work. Compared to previous solutions only slight adaptations are needed to fulfill their specific needs. Furthermore, personalization settings can be stored in the common database of the system. This grants that users have the same individual user interface available after logging in, regardless of the workstation they use.

2.5 Adaptability of ERP Systems

When developing an ERP system it is of great importance to identify and specify the business requirements of a company since these requirements are reflected in the business processes that are conducted and supported by the ERP. In the case of standard software such processes are adopted for a variety of companies based on best practice reference models since they show several similarities from enterprise to enterprise as described in Chapter 2.3. In order to fulfill these predefined requirements as best and efficient as possible, ERP systems are further customized and personalized (cf. Chapter 2.4).

Business requirements can not be regarded as definite characteristics since market demands can change over time. Enterprises therefore have to occasionally question the stability of their processes. Sometimes conducting short-term reactions to changed needs is necessary, which requires companies to be very flexible. The ability of adjusting to new conditions fast and efficient can be a critical factor for the success of an enterprise. Due to that reasons, a versatile enterprise architecture is required, where business processes, the information architecture, and the application landscape are closely coupled [24].

When changes in the business processes occur, application systems that are used by the company are also replaced when they are not capable of fulfilling the changed needs. Especially ERP systems, that support a predefined set of resources and core processes, are often not flexible enough to do so. Changes of business needs can often only be represented in an incomplete and inefficient way, by changing the data structure or the code of the ERP systems. But such drastic changes should be avoided since inconsistencies could be the consequence. ERP systems therefore also need to be more adaptable and flexible with respect to future changes in the business. Customizing is not sufficient since only processes that were predefined by the vendors of standard software are affected.

Adaptable ERP systems overcome this pitfall and are capable of adjusting to changed needs. Gronau [24] distinguishes two dimensions of adaptability: *technical* and *enterprise-specific*. Technical adaptability describes the potential of a system to adapt to changes. Indicators for that characteristic are *interoperability*, *scalability*, *modularity*, *availability*, *independence*, *self-organization*, and *self-alikeness*. **Scalability** describes how a system can be designed in form of capacity. This means that the system is able to efficiently adjust to an increasing or decreasing amount of processed data, which can happen in form of hardware- or software adaptations. Adding or removing storage- or computing performance should enable a flawless operation of the system. The architecture of an ERP system should be structured in small autonomous subsystems (or modules) and therefore be **modular**. These modules consist of interfaces describing their functionality and a part implementing it. Because of these interfaces, modules can be removed and substituted by modules with the same functionality quite easily. Modularity therefore facilitates the efficient reuse, combination, and adaptability of applications. **Availability** depicts the need for ERP systems that can be used from everywhere and every time. Platform **independence** is also important. Furthermore, system failures should not have negative effects on other systems. **Interoperability** describes how applications are capable of collaborating with each other. A better intra- and inter operability can be achieved by following standards when planning, implementing, and using application systems. The ability of a system to adjust its internal structure by self-regulating mechanisms that are based on interactions with its environment (e.g., user inputs), is called **self-organization**. **Self-alike** systems should look similar and show a reoccurring operating philosophy, which eases the initial training effort for users. Compared to technical adaptability, enterprise-specific adaptability specifies how well a system is able to reflect changes in underlying business processes and to what extent it is possible to reconfigure the system at runtime [24].

The adaptability of ERP systems can be increased by a variety of approaches and actions. Gronau [24] names the *model-view-controller* software pattern, *component based architectures*, and *service oriented architectures* as examples. He further argues that business process modeling should be integrated in the systems to create or adapt an ERP and mentions modeling languages like *Business Process Model and Notation* (BPMN) [27] as example. Instead of using business process modeling techniques, business models might be a better choice to illustrate the business needs of a company. Since not all tasks have to be specified in detail, these models can be designed and interpreted more easily as described in the following chapter.

CHAPTER 3

The REA Ontology

3.1 Business Models

As explained in Chapter 2, a company's ERP system can either be developed by its own IT-department or by external companies. Besides that, already existing ERP solutions of small vendors or big companies like SAP that need to be customized to the customers specific needs are an option. Regardless of the chosen way to implement an ERP, its underlying data structure and the user interface have to reflect the economic phenomena that build the foundation for the business of the company using the system [41]. Otherwise it will be impossible to create meaningful ERP systems and guarantee a productive use.

In order to create a data structure that is capable of reflecting the economic phenomena of a business, business people have to be able to clearly formulate their needs and communicate these requirements to IT-specialists that are implementing the system. Unfortunately this task is not trivial since entrepreneurs or managers naturally understand how their business works, but struggle when trying to explain it in a simple and unambiguous way.

Business models can increase the mutual understanding between the business and IT domain [57] (cf. Figure 3.1 on page 24). They can contribute to requirements engineering of information systems by gathering and representing high-level business goals [11]. Using a business model as common language, business people can ensure that IT-professionals understand what economic phenomena need to be represented in the information system and its data structure. Giving a precise description of a business model is hard since research is mainly conducted in silos. Even scholars do not agree on a consistent specification of the concept and adopt definitions fitting their own research purpose [70]. Osterwalder et al. [57] describe a business model as a conceptual tool that expresses the business logic of a specific firm by concepts and their relationships to represent what value is provided to customers. Timmers [64] defines it as architecture for the product, service-, and information flows, including a description of business actors and their roles, a description of potential benefits for the actors, and a description of the sources of revenue. According to Andersson [1] business models specify the main actors of the business, their relationships, and the values that are exchanged between them. Zott et al. [69] name them a system of interdependent activities that transcends the focal firm and spans its boundaries. Magretta [40] entitles them as stories that explain how enterprises work and answer questions like: Who is the customer? What is valuable to the customer? How does the company make money? What is the underlying logic explaining how value is delivered to the customer? Another popular definition is the one by Rappa [59], who describes a business model as the method of doing business by which a company can generate revenue. He argues that the model states how companies make money by specifying its position in the value chain. These definitions differ a lot, but as Zott et al. [70] argue there are some reoccurring themes. Business models use a holistic approach to describe how a company does business and try to explain how value is created and captured. Furthermore, in business models not only the activities of the focal firm are of importance, but also activities of partners that exchange value with the company.

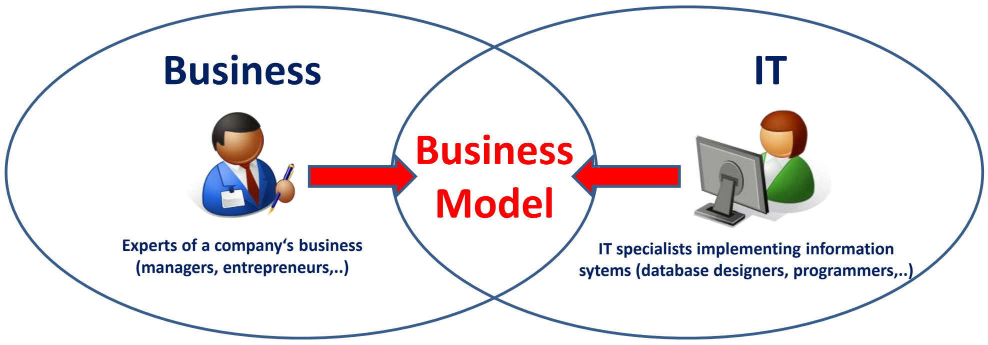

Business
IT
Business Model
Experts of a company's business (managers, entrepreneurs,..)
IT specialists implementing information sytems (database designers, programmers,..)

Figure 3.1: Business Models as common Language for Business and IT

Business models have to be clearly distinguished from *business process models*. As stated before the concept of *value* seems to be important in business models. Gordijn et al. [22] share this opinion and point out that business models should describe who the value adding business actors are, what elements these actors are offering to each other in return, what value-creating or value-adding activities are producing or consuming the offerings, and which of these activities are performed by which actors. In other words a business model describes *who* is offering *what* to *whom* in return for *what* during *which* value creating activity, but they don't explain *how* this happens in detail. As mentioned in Chapter 2.1, *business processes* are partially ordered sets of tasks or steps undertaken towards a specific goal [9]. The activity of constructing models that represent such processes is called *business process modeling* (BPM) and can be used to create a common approach for work that needs to be done or to improve existing processes [19]. Business process modeling can therefore be used to describe *how* the aforementioned activities are carried out [22]. Summarizing, a business model can be seen as a representation of a companies logic to create value and exchange it with other actors. A business process model on the other hand defines how concrete business cases are implemented in processes [57]. For the modeling of business processes several tools and languages exist like the IDEF family, the business process modeling notation (BPMN), petri nets, event process chains (EPC), or UML activity diagrams as described in [53]. In the context of business model conceptualization the Business Model Ontology (BMO) [55], e^3-value [20], and the Resources-Events-Agents (REA) ontology [49] can be named as the most popular examples. Following, we dwell on these different business models.

The Business Model Ontology (BMO)

BMO is an ontology that precisely allows to describe the business model of a firm. Comparable to the four perspectives in the balanced scorecard approach (financial, customer, internal business processes, learning and growth) [37], its concept is based on pillars representing facets that are essential to a company's business: *product*, *customer interface*, *infrastructure management*, and *financial aspects* [55]. These pillars are further divided into nine building blocks that represent concepts of business models that were mentioned by several authors in their work (cf. Table 3.1). A more thorough description of these blocks can be found in [56].

Pillar	**Business Model Building Block**	**Description**
Product	**(1) Value Proposition**	Overall view of a company's products and services that are of value to the customer.
Customer Interface	**(2)Target Customer**	The segment of customers a company wants to offer value to.
	(3) Distribution Channel	Defines how companies can reach their customers.
	(4) Relationship	The kind of link a company establishes between itself and the customer.
Infrastructure Management	**(5) Value Configuration**	Describes the arrangement of activities and resources that are necessary to create value for the customer.
	(6) Capability	The ability to execute a repeatable pattern of actions that is necessary in order to create value for the customer.
	(7) Partnership	A voluntarily initiated cooperative agreement between two or more companies in order to create value for the customer.
Financial Aspects	**(8) Cost Structure**	Representation in money of all the means employed in the business model.
	(9) Revenue Model	Describes the way a company makes money through a variety of revenue flows.

Table 3.1: Four BMO Pillars and their respective Building Blocks [55]

Identifying these nine building blocks for a specific company helps to detect *what* products are valuable to the customers (1), *who* the customers are (2), *how* these customers are reached and *what* relationships to them exist (3)-(4), *what* is needed to create value for the customers (5)-(6), *which* firms cooperate with the own company (7), *what* the cost structure is (8), and *how* the company makes money (9). BMO can therefore assist domain experts when creating the business model, but it does not conceptualize the elements that occur in the model.

e^3-value

e^3-value was developed to conceptualize e-business models. Before its introduction many interpretations existed, increasing the risk of misunderstandings and failure of the models. The focus of e^3-value lies on the concept of value. Domain experts can model how actors in a network create, exchange, and consume valuable objects [20, 21]. e^3-value has an easy to understand graphical syntax and comes with an editor that makes the creation of models rather simple[1].

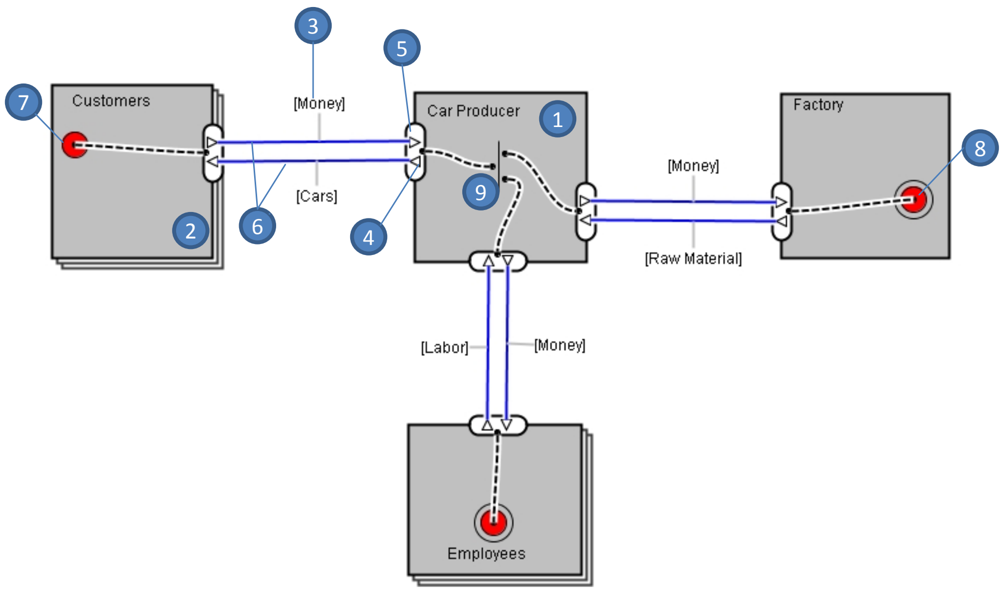


Figure 3.2: e^3-value Model of a Car Producer

Figure 3.2 shows an e^3-value model of a car producer who interacts with other actors in a network and exchanges valuable objects. The model consists of several concepts. The car producer itself and the factory are **actors**, illustrated as rectangles (1). Customers and Employees are represented as stacked rectangles depicting them as actors belonging to **market segments** (2). Between actors **value objects** (3) are exchanged. Examples are money and cars. Actors show what value object they are requesting or provisioning using **value ports** (4). These value ports are grouped together by **value interfaces** (5) that model *economic reciprocity*. This means that value objects are not given away for free, actors expect something in return. In Figure 3.2 the car producer sells cars in return for money and pays money to buy raw material. **Value Transfers** (6) connect two value ports and illustrate trades of value objects that are possible. Scenario paths start with a **start stimulus** (7) and end with an **end stimulus** (8). The fact that the money from the car sales is used for buying raw material as well as acquiring labor is illustrated by the **AND fork** (9). After the creation of the model, e^3-value enables domain experts to assess its economic sustainability in form of profitability sheets that can be created automatically. Guidelines on creating value models from the scratch can be found in [21].

[1]`http://e3value.few.vu.nl/tools/`

A disadvantage of e^3-value is that commitments and policies, both important concepts in ERP systems, are not supported. Furthermore, actors only represent companies. Internal processes can be depicted by **value activities** (not shown in Figure 3.2), but the company-intern employees participating in those activities cannot be represented.

REA

Compared to the aforementioned business ontologies, REA has all the characteristics that are preferable in order to build a data structure on it. Commitments are supported as well as policies, model elements are clearly conceptualized, and its roots lie in the accounting discipline. Additionally it was proposed to support the implementation of IT-Systems and is related to the subject of data modeling. Due to that reason the database of the REAlist project is based on REA as explained in Chapter 5. Since the concept should be well understood, the rest of this chapter is entirely dedicated to REA.

3.2 History of the Resource-Event-Agent (REA) Framework

In order to integrate ideas of accounting theory and database systems, McCarthy [47] already proposed in 1979 to use Chen's [4] entity-relationship concept to create database models for accounting systems without proposing the usage of specific database systems. McCarthy argues that in a database environment accounting systems are modeled as real-world entities and relationships among them. As an example an accounting system for an enterprise is modeled. Instead of using traditional accounting concepts like a chart of accounts or double-entry accounting procedures, an entity-relationship model is used. In order to create the model McCarthy at first identifies the sets of objects (later on resources), agents, and events that exist in the enterprise. Furthermore, the relationships that connect these entities are identified. Based on this sets the entity-relationship model for the accounting system is built. Relationships between two events represent manifestations of double-entry accounting conventions which means that each change in the resource set leads to a change in the related entity. McCarthy shows that, although the developed entity-relationship data model differs from the traditional accounting paradigm (chart of accounts and double-entry bookkeeping), such systems can accommodate conventions of traditional accounting [47]. It is interesting to notice that despite the fact that the term Resource-Event-Agent (REA) was not intended so far, McCarthy already envisioned agents, events and objects as entity sets.

In 1980 McCarthy [48] criticized the traditional accounting model. One point of criticism is that the classification schemes for data concerning economic transactions and objects, based on the chart of accounts, are not appropriate. Another point is the limited dimension since accounting measurements are primarily expressed in monetary terms. The integration with other areas of an enterprise is also mentioned as a drawback since information concerning the same entities is maintained separately by different people. McCarthy builds a basis for reevaluating traditional accounting concepts. He therefore uses the aforementioned entity-relationship modeling techniques for accounting systems. Compared to the conventional framework, elements can also include non-accounting events and can have non-monetary characteristics. Again, entity-relationship sets are used instead of journals and ledgers. As mentioned before, the entity-relationship methodology starts by identifying existing entities in the modeled reality and relationships among them. In this context the entity groups building accounting object systems are identified as (1) *economic events*, (2) *economic resources*, and (3) *economic agents*. The term “economic“ will often be omitted in the remainder of this thesis to ensure better readability. McCarthy also argues that the entity-relationship accounting model enables non-accountants to add characteristics or entity sets more easily than the conventional accounting model [48].

3.3 The basic REA Framework

Based on his previous work McCarthy finally introduced the actual REA Accounting Model in 1982 [49]. REA was developed as a generalized accounting framework that can be used during the database design process of accounting systems. Consequently, accounting can be included as part of an enterprises database system that satisfies the needs of accountants as well as non-accountants over shared access to an enterprise schema. The central unit of analysis are economic exchanges. Due to McCarthy’s earlier work [47, 48], these exchanges are represented by more intuitive concepts than debits, credits, and accounts. The reason for that is that these artifacts are associated with journals and ledgers. In an accounting system, these concepts are not essential. The things that are of interest to accountants are *monetary stocks* of an enterprises goods or claims at a specific time. Furthermore, the *monetary flows* of these items that occur are important. The stocks correspond to the balance sheet accounts of the general ledger whereas the flows correspond to the income statement accounts. Therefore accounting phenomena can be divided in two categories: *stock objects* and *flow transactions*. As already mentioned McCarthy introduced a categorization of accounting phenomena into resources (or objects), events, agents and the associations between them in [47]. The REA framework generalizes these concepts in order to reflect the stock-flow aspects of accounting systems [49].

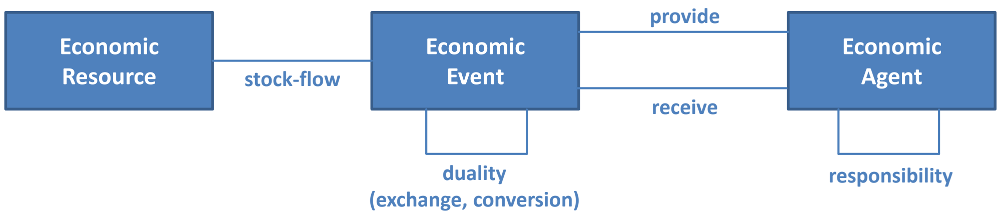


Figure 3.3: REA Accounting Model based on [32, 49]

Figure 3.3 shows the basic REA accounting model. It consists of sets representing economic resources (R), economic events (E), economic agents (A), stock-flows that illustrate which resource is incremented or decremented during which event, and participate relationships, depicting which agents provide or receive resources. Before describing these sets and relationships in greater detail, the meaning of a process in the REA context has to be clarified. Following the definition of Geerts and McCarthy [16], in an economic sense, a process is a production function where the entrepreneur exchanges specific input resources for some output resources that are of greater value to the customer. This definition differs from the explanation that was given in

Chapter 3.1, defining processes as partially ordered sets of tasks. A REA business model does not necessarily contain descriptions of the detailed tasks that need to be carried out during an event. Nevertheless it is possible to do so and use REA as business process modeling language as will be explained in Chapter 3.4. Similar to Geerts et al. [14], we often use the terms exchange, process, and activity to mean the same thing. In this chapter the term process will be used to describe compensating REA events, but in the remainder of this thesis we will use the term business activity to clearly distinguish them from the classic definition of business processes.

Economic Resources are scarce objects or items. As Ijiri [35] explained, these objects are important to an enterprise and are controlled by it. Examples for resources are goods, rights, or services that need to be monitored and controlled due to their high relevance for enterprises.

Economic Events are phenomena that represent changes in these scarce objects (economic resources). These changes can result from activities like production, exchange, consumption, or distribution [67] that consist of at least two events. This fact is illustrated by the **duality** relationship. Following the definitions of Ijiri [35], these relationships are described as the linkage between the *increments* in the resource set of an enterprise and the corresponding *decrements*. The increments and decrements are members of two different event entity sets. One of these two entity sets is transferring in a resource. The other set is transferring out a resource. In other words, the duality relationship combines distinct events to an economic process. An example for such a process might be the buying of goods where a purchase event and a cash disbursement event are connected. The purchase leads to an inventory increase. But since nothing is for free, something has to be offered in exchange in accordance to the give-take principle. Therefore the related cash disbursement event leads to a decrease of cash.

Economic Agents participate in events and therefore in the economic processes. Examples for agents are persons, organizational units, or companies that have control of resources. These resources are exchanged between agents during processes. McCarthy [49] distinguishes between parties inside and outside of the company. In that sense inside agents are agents that work inside the enterprise being accounted for (e.g., employees like salespersons, cashiers, or shop assistants). They can be responsible for the participation of subordinates as indicated by the **responsibility** relationship. Examples for outside agents are vendors and customers, that participate in a process but do not belong to the company.

Hruby [32] distinguishes two different forms of economic processes: *exchange* and *conversion*. As illustrated in Figure 3.3 dualities can either be **exchange-dualities** or **conversion-dualities** depending on the processtype. An example of an exchange process is a purchase where cash is exchanged for inventory. Exchanges occur between inside agents and outside agents (buying inventory from a vendor). Contrary, during a conversion process resources are changed in form or substance in order to generate greater value inside the company. An example for a conversion process is pizza production, where ingredients are used to create pizza.

The remaining relationships that need to be described are **stock-flow**, **provide**, and **receive** (the latter two are summarized as **participation** relationships [49]). A Stock-flow relationship connects economic resources to economic events. It therefore defines what (resource) is exchanged (or used) when (event). Participation relationships relate an increment or decrement event to an agent. Depending on the events agents receive resources during an event, or provide them.

In its beginnings the REA Accounting Model was used as a framework to analyze the information needs of an enterprise. Starting with an arbitrary identified resource that is of interest to the enterprise, the designer of the database knows that two event sets are needed. One of those representing an inflow, the other one an outflow of the resource. In each of these event sets economic agents participate as defined in the model. Using this method database designers and accountants can specify a conceptual schema of the enterprise.

Figure 3.4 shows an example of an exchange process instance. Again the car producing company, whose flow of valuable objects was modeled using e^3-value techniques (cf. Figure 3.2 on page 26), is used for illustration purpose. The process is modeled from the view of a car producer who makes money by selling cars to customers. Therefore the resource `Audi A4` is provided by the `inside agent Hank` during the `sale decrement event` on 30th July 2013 to the `outside agent Walt`. In exchange for the received car the customer provides `10.000 €` to the salesperson during the `cash receipt increment event` on 31st July 2013. This give-take relationship between the two events is emphasized by the `duality` relationship. Since the car producing company gives up a resource to gain another resource, the sales process is an exchange. Resources, events, and agents are all represented by specific instances since the events have already happened or are happening right now. The sales process illustrates a concrete business case.

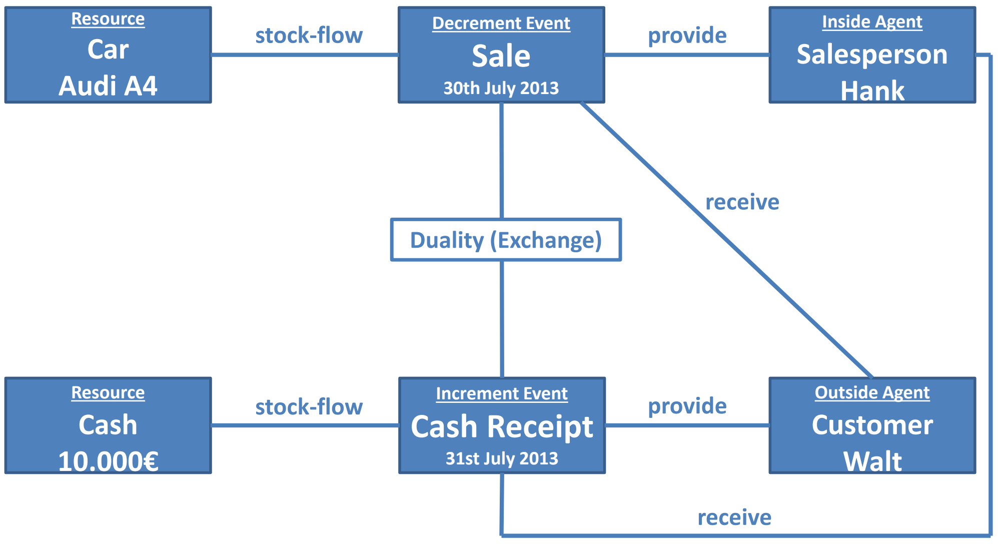


Figure 3.4: REA Sales Process of a Car Producer with Instance Data

3.4 Engineering Business Processes and Value Chains using REA

As explained in [16] the usage of REA was expanded from designing accounting information systems to modeling economic phenomena in general. Therefore REA can be used to model business processes. In an Enterprise Information System not only one process, where someone gives up control of a resource in order to gain control of another resource, is of interest. Several processes are forming an enterprise value chain as defined by Porter [58].

Since the resource outcome of one process can be used as input for another process, the aforementioned stock-flow relationships combine processes to value chains as described by Geerts and McCarthy [14]. In that case incremented resources that are the outcome of preceding processes are used as input resources that are decremented in a succeeding process. An example value chain of the car producer is shown in Figure 3.5. The value chain consists of four processes: *sales*, *purchase*, *labor acquisition*, and *car production*. Inflowing and outflowing resources of these processes are *cash*, *labor*, *raw material*, and *cars*. During the sales process cars are sold for cash. This cash is used in the purchase process to buy raw material. Compared to these two processes, where one resource is exchanged for another resource, the production of cars and the labor acquisition are examples for a conversion. Cash is used to produce labor. This labor is used and raw material is consumed to produce cars. Agents are omitted in the representation of the value chain. Each one of the aforementioned processes can be represented in a more detailed way using the concepts described in Chapter 3.3. Figure 3.4 on page 30 shows the detailed representation of the sales process.

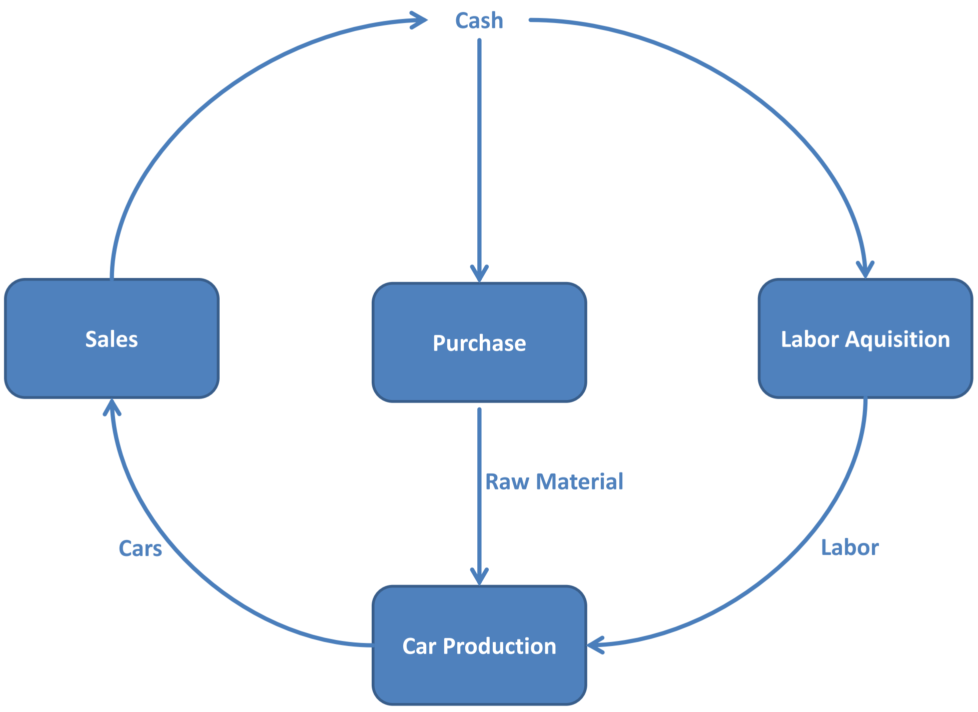


Figure 3.5: REA Value Chain of a Car Producer

Besides the value chain specifications on enterprise level and the process descriptions based on REA, Geerts et al. [16] introduced the task-level as third layer. They argue that a full REA model often is not technologically attainable or useful and introduce tasks as a more detailed segmentation that are needed to accomplish events. Compared to events, tasks don't need to be planed, designed, monitored, or evaluated. Furthermore, they cannot be paired with a counterpart like events using the duality relationship. Tasks are individual steps that need to be carried out during an event. The possibility of defining tasks makes REA a process modeling language according to the definitions stated in Chapter 3.1.

3.5 Advancement to the REA Ontology

Gruber [28] describes an ontology as an "explicit specification of conceptualization". While *upper-level ontologies* define concepts shared by all domains, *domain ontologies* define things that are only relevant to a specific domain [15]. Business ontologies therefore define things and phenomena that occur in enterprises and businesses. In order to reuse accounting concepts in different applications and systems, Geerts et al. [13] tried to make knowledge intensive use of the REA framework and named it as candidate for a full domain ontology. REA can be used to specify the conceptual schema of an enterprise based on occuring economic phenomena as described in Chapter 3.3. Due to the fact that this perfectly fits the ontology description by Gruber and REA's foundations in the economic/accounting theory, extending REA to an business ontology made sense.

Using REA to conceptualize economic phenomena underlies some restrictions that always have to hold. Geerts and McCarthy [15] therefore defined three axioms to specify these rules:

- **Axiom 1 -** At least one increment event and one decrement event exists for each economic resource in order to guarantee the modeling of a companies economic activities as a sequence of exchanges.
- **Axiom 2 -** Events that effect an outflow of a resource must be paired with events affecting an inflow and vice-versa. This concept has already been explained before using the term "duality".
- **Axiom 3 -** In an exchange inside agents and one outside agent participate, ensuring that resources are exchanged between parties that have competing economic interests.

Taking into account the process distinctions proposed by Hruby [32], these axioms only hold for exchange processes. For conversion processes the rules regarding the number of participating agents differ. The same agent who provides resources that are used or consumed during the process can receive the produced resources. Furthermore, only inside agents participate in a conversion [33].

Apart from the axioms for REA modeling, Geerts et al. [15] proposed to extend REA *vertically* and *horizontally*. The vertical extensions have already been discussed in Chapter 3.4, where enterprise value chains and tasks were introduced besides the REA duality concept. Horizontal extensions include the definition of an *operational layer* and a *knowledge layer* (also termed *policy layer*). While the operational layer uses the basic REA concepts to specify what is actually happening or will happen in the near future, the policy layer specifies constraints or guidelines for business operations. Modelers can therefore define what could, should, or must be [18]. The horizontal layers enable the analysis of economic activities at different points in time. This can be done for value chains, duality specifications, as well as tasks. Figure 3.6 on page 33 shows the extended REA ontology.

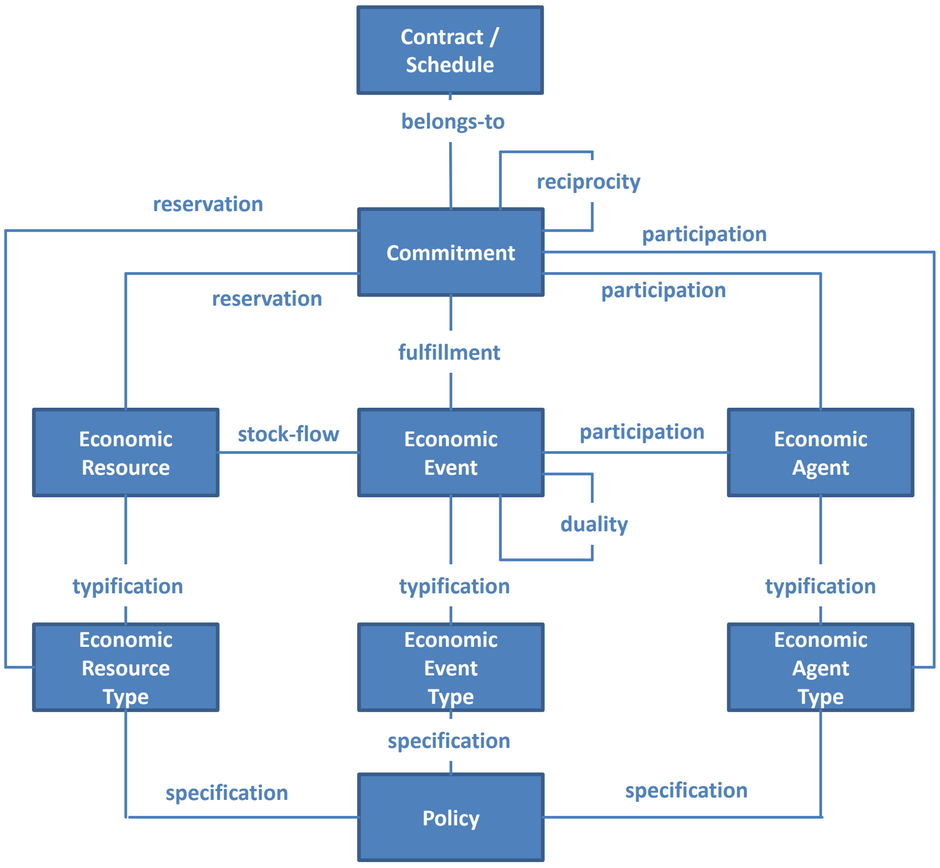


Figure 3.6: REA Ontology [32] (adapted)

The essential concepts introduced on the policy layer are *types*. Types are abstractions that represent groups of entities [17] and describe intangible components of actual phenomena [15]. Types exist for all of the aforementioned entities (**agenttypes**, **resourcetypes**, and **eventtypes**). The **typification** relationship links agents, events, or resources to their corresponding type entities. Relationships between types can be defined to specify restrictions or guidelines. Geerts et al. [18] name five patterns that are useful to model such **policies**. An example of such a restriction is the relationship between two types specifying what kind of plane can be used for what kind of flight. This policy has to be fulfilled by all instances of the type entities (flights and planes). Furthermore, such relationships can also be defined between types and actual instances. For example it therefore is possible to describe that an eventtype demands the participation of a specific agent.

On the operational layer **commitments** were added. While events describe something that is actually happening, commitments are an agreement to execute an event in the well-defined future. This fact is represented by the **fulfillment** relationship in Figure 3.6, that determines which event will be executed to fulfill the commitment. Like events, commitments also occur in combination with at least one other commitment. The reason for that is that a commitment to give up a resource is always accompanied by a promise to receive another resource instead. The relationship between two commitments is called **reciprocity** and is described at a higher level by the concept of **agreements**. Two types of agreements exist: **contracts** and **schedules**. While contracts are a collection of commitments promising exchanges between agents, schedules are a collection of conversion commitments. An example of a contract is a sales order consisting of various sales. Schedules can be used to identify resources and plan their use in production orders consisting of several production jobs [32]. Commitments **involve** agents and **reserve** resources or resource types that will be used when events fulfill the commitments.

Custody and **linkage** are introduced as new relationships. Resources can consist of several parts but since these parts can also be resources, a new entity in the model is not optimal. The linkage relationship describes dependencies between resources as part-whole relationship. For example the resource car consists of parts like wheels, doors, or windows. The specification of such resource parts can be useful for scheduling conversion processes. Custody relates agents to resources. Therefore it is possible to define which agent is responsible for what resource and who can be contacted if a problem with the resource occurs. Examples are warehouse clerks responsible for the material in the warehouse or cashiers responsible for the cash register. Custody, linkage, and responsibility can also occur between type entities [32]. This fact is represented by the **specification** relationship in Figure 3.6.

Due to these extensions of the basic REA framework, REA can be considered as a powerful business ontology. It is capable of representing all data that is relevant for the conceptual design of ERP systems [42].

Figure 3.7 shows the planned form of the sales process of the car producer with some extensions. Compared to the process in Figure 3.4 on page 30, the resource exchange does not happen right now but is planned instead. The `sales commitment` plans that `salesman Hank` and a `shop assistant` will provide the resource `Audi A3` to the `customer Walt`. It therefore `reserves` the resource for the occurring transaction. The concrete shop assistant that will participate in the event is not specified yet. Every shop assistant in the company can take part. Therefore the agenttype is related to the commitment. The `cash receipt commitment` plans the cash income and `reserves` the money that will be paid by the `customer Walt` to the car producers `cashier Rachel` during the exchange. The commitments are in a `reci-procity` relationship. Together the two commitments build a `sales order contract`. The actual transaction will happen in the `sale event` on 30th July 2013 and the `cash re-ceipt event` on 31st July 2013. The same four agents that were planned will participate in the events that `fulfill` the corresponding commitments.

3.6 REA in Information Systems

David et al. [62] propose an REA information architecture as a normative model. This model is used to compare ERP systems among each other and should give better indications on their

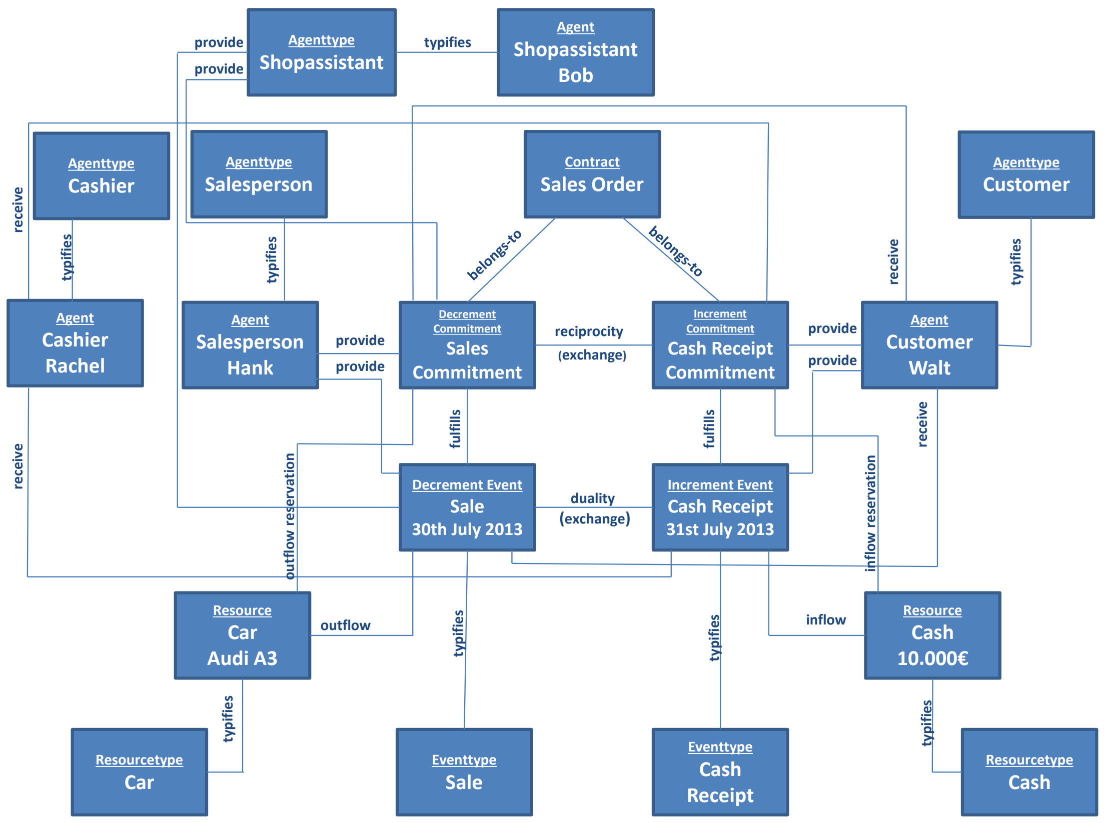


Figure 3.7: Planned REA Sales Process of the Car Producer

quality than a comparison at the implementation level. However, REA as basis for the data structure of an ERP system is not mentioned.

O'Leary [54] investigates the relationships between the underlying data models in REA and the dominant ERP system SAP. The findings illustrate that although several similarities exist, SAP has some implementation compromises that prevent it from being fully REA.

Rosli et al. [60] applied the Resource-Event-Agent (REA) data model to specify and design an accounting information system. A prototype based on the REA data model was implemented. Compared to the REAlist project no database capable of storing all sorts of REA processes is used. The prototype is limited to the revenue cycle scope.

There are several projects where REA was used as a data model for an information system but the concept of the partly generic database as proposed in the REAlist project was never applied. Furthermore, the approach of using a graphical modeling language to define and adapt REA processes, storing these models in the database, and using them for ERP adaptation is also new.

CHAPTER 4

The REA-DSL

4.1 A dedicated REA Representation Format

As explained in Chapter 3, REA is a business modeling ontology that can be used to describe important economic phenomena for the conceptual design of ERP systems. It enables the creation of models by defining economic events that occur within and across enterprises using the concept of value chains. Besides the representation of events that have already happened or are happening, events that are scheduled or planned by commitments are important. Especially when designing Accounting Information Systems (AIS) or ERP systems such things are crucial since they are important tools for managers to not only keep track of past events but also to predict the financial future of companies and take proper actions [41]. REA therefore seems to be a good choice for defining business models and building the data structure of information systems on it.

Unfortunately using the class-diagram-like representation of REA is very vague and therefore leaves space for misinterpretation. No multiplicities are defined on the relationships between entities like resources, events, agents, or commitments. It is not clear how many resources can be linked to events with stock-flow relationships, and the number of agents participating in events is also undefined. Furthermore, it is not described what happens when resources that are given away during an economic exchange are connected to increment events (what should clearly not be the case). Domain experts therefore have to check created REA models in a very detailed way. Such a task is especially cumbersome since REA models in class-diagram-like representation can be very complex. Even a simple exchange process already consists of an increment event, a decrement event, two resources, and two agents (cf. Figure 3.4 on page 30). When type entities are included and commitments are occurring, the model grows even larger and more confusing since all entities are represented in the same form as rectangles. Due to the missing dedicated representation format of REA, users have problems defining models in an easy and understandable way [63]. Therefore, its usage as common communication language between business experts and IT-professionals during the design phase of Accounting Information Systems might lead to unsatisfying results.

The REA-DSL tries to overcome these limitations by defining a dedicated graphical representation for the REA ontology. As the name intends, the REA-DSL is a domain-specific language (DSL). Compared to general purpose languages like C, Java, or Python, DSLs are focused on (and usually restricted to) particular domains and therefore cannot be applied to an arbitrary problem. They can be designed to solve technical tasks as well as business tasks and their power lies in their expressiveness for the domain at hand (in case of the REA-DSL: modeling of REA business models). This is achieved through appropriate notations and abstractions for the domain the DSL is tailored to [65]. DSLs are developed for users that are familiar with the problem domain (domain experts). An example might be an accountant well aware of the business activities occurring in an enterprise who uses the DSL to create REA business models. Solutions can be expressed at an abstraction level of the problem domain, enabling domain experts to understand, validate, and modify the artifacts created by the DSL more easily [45]. Developing graphical domain specific languages for REA is a popular topic. Sedbrook [61] also specifies a DSL meta-model and a prototype to create visual business models that conform to the REA ontology. Similar to Mayrhofers approach, user interfaces to design operational- and policy level models are provided. Using code generation techniques Sedbrook also transforms the designed models into executable code that can support business applications.

During the development of the REA-DSL the approach introduced by Strembeck and Zdun [68], consisting of four main activities (*core language model definition*, *DSL behavior definition*, *concrete syntax definition* and *platform integration*), was used. The DSL was extracted from an already existing system (the REA ontology). Therefore the core language model was created by identifying the elements of the REA ontology (resources, events, agents, commitments,..), relationships among them (stock-flow, participation,..), and defining constraints on that language model (as indicated by the ontology constraints on page 32). The resulting meta-model was built based on the multilevel modeling architecture Meta-Object Facility (MOF) as introduced by the Object Management Group (OMG) in [26]. On its highest level (M3) MOF defines concepts that can be used to create meta-models. Doing so lead to the development of three interlinked views in the REA-DSL's initial form: *agent view*, *resource view*, and *operational view* [45, 63]. This model was later on extended by the *value chain view* and the *planning view* in order to capture most of the REA ontology's concepts [43]. Brief descriptions of the concepts described in each view are given. A more thorough visualization of the meta-models is illustrated in [41]. The agent view describes the concepts of agents, agenttypes, the typifications between them, and generalization hierarchies. In the resource view resources, resourcetypes as well as typifications and generalizations between those concepts are included. Furthermore, the concepts of *labor resources* and *bulk resources* are introduced. As Mayrhofer [41] argues, labor is a special kind of resource that occurs in almost every event. Bulk resources are resources that cannot be identified individually or where individually tracking them makes no sense. Examples for bulk resources are nails or water where only the quantity on hand is of interest. In the REA-DSL bulk resources are a special kind of resourcetype since they also define characteristics that are common to a set of resources. The rules for defining duality relationships where one event is compensated by another event is specified in the operational view meta-model. Sonnenberg et al. already pointed out in [63] that a duality does not only encompass one decrement event and one corresponding increment event. Rather a set of increment entities can be compensated by a set of decrement entities. The number of agents participating per event and resources that are incremented or decremented is formalized in the operational view meta-model as well. The value chain view corresponds to the concept of value chains as described in Chapter 3.4 and defines how many processes can occur in a value chain. Furthermore, the number of possible inflows or outflows of resources/resourcetypes is defined. Finally, the planning view meta-model builds the basis for the definition of the commitment concept and defines the rules for valid relationships between all included entities like commitments, the events fulfilling them, reserved resources/resourcetypes, and involved agents.

The behavior of the DSL was defined by analyzing how the language elements interact with each other to produce the intended behavior. Concrete syntax can either be textual or graphical. The REA-DSL provides a graphical syntax in order to define a dedicated representation of the REA ontology and also includes a modeling tool support for the DSL. This syntax, based on the five aforementioned views, is discussed next.

4.2 Graphical Syntax of the REA-DSL

As mentioned before, the intention of the REA-DSL is to define a dedicated graphical representation for the REA ontology. Based on the meta-models of the resource view, agent view, operational view, value chain view, and the planning view a graphical syntax was created. Compared to the class-diagram-like representation of REA models, the notation elements of the REA-DSL are intuitive and simple and therefore easier to use for domain experts when creating business models.

In order to explain the graphical syntax and use of the DSL as simple as possible the example of the car producer will be used. The value chain, showing all value adding activities of the company, was already stated in Chapter 3.4 (cf. Figure 3.5 on page 31). The car producer sells cars to its customers to make money. This money is used to buy material for the car production from a factory. Furthermore, the employees have to be paid. Four activities can be identified: *sales*, *purchase* (which are exchange activities), *labor acquisition*, and *car production* (which are both conversion activities). The resources flowing between the activities are also given: *cash*, *raw material*, *labor*, and produced *cars*. Based on this information, users of the REA-DSL can start to define the resources that are exchanged, used, consumed, or produced during the value adding activities in the value chain.

Resource View

Using the REA-DSL, the aforementioned resources that occur in the value adding activities of the car producer are modeled as drops. Solid drops represent resources that are individually identifiable. Dashed drops represent resourcetypes or bulk resources where no tracking of individual instances makes sense. The REA-DSL representation of the car producers resources is shown in Figure 4.1.

`Labor` is, as already mentioned, a special kind of resource. Therefore it is not represented as a solid/dashed drop as other resources but includes three stick figures inside the drop. According

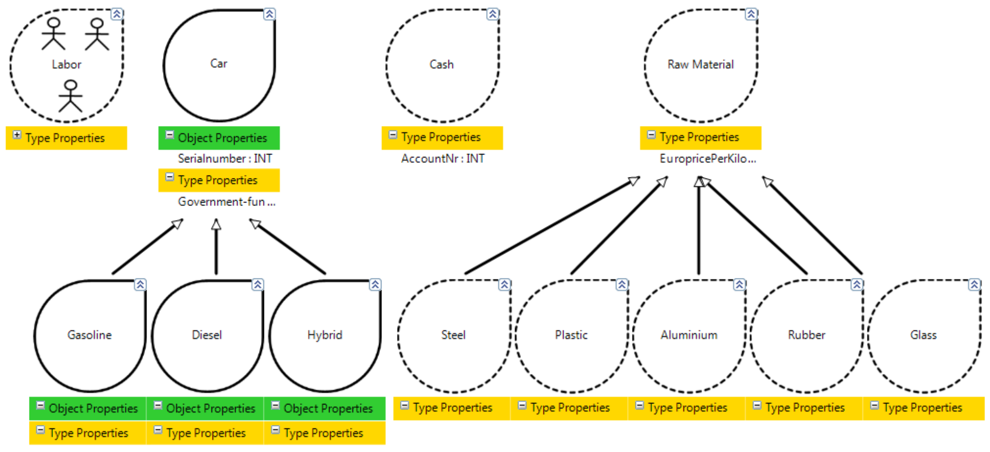


Figure 4.1: REA-DSL Resources

to [41], `labor` is acquired during the payroll activity. It defines all the minutes agents participate (and therefore provide work) in events.

`Car` is represented as a solid drop and therefore is an identifiable resource. As indicated by the generalization hierarchy it has three subtypes: `gasoline`, `diesel`, and `hybrid`. Sub resources are specializations of their super-resources and inherit their attributes.

`Cash` and `raw material` are bulk resources that cannot be individually identified. Not every coin, banknote, or raw material is tracked. Instead the amount of types of those resources is recorded. `Steel`, `plastic`, `aluminium`, `rubber`, and `glass` are specializations of `raw material`. These resources are represented as dashed drops in the same way as bulk resources.

An important thing that can be represented using the REA-DSL are properties as indicated by the green and yellow sections beneath each resource/resourcetype. Object properties are visualized in green and are properties that differ from object to object (affecting the operational layer). For example the `serialnumber` property of each individual car is unique. Type properties (represented in yellow and affecting the policy layer) on the other hand stay the same for types of objects. E.g. all types of hybrid cars have the same `government_funded` value (hybrid cars are funded by the government since they are environmentally friendly). These properties lead to a further distinction of resources that was proposed in the course of this thesis. When a resource is identifiable but has object- and type properties, it is regarded as *identifiable resource*. Such a resource can be tracked on its own but also shares common characteristics with other resources like the aforementioned cars. A *unique resource* only has object properties. An example is a painting of a famous painter that only was created once and therefore is unique. Compared to unique resources, identifiable resources can be reordered by customers. Resources and resourcetypes also have fixed attributes like their `name`. Additionally bulk resources have two more fixed attributes `QoH data type` and `QoH unit`[1]. These attributes are important for resources that cannot be tracked individually. While `QoH unit` defines the unit of the measured

[1]QoH stands for quantity on hand

resource (e.g., pounds of raw material), `QoH data type` is the data type of the resources (e.g., double for the amount of pounds).

The reason for the introduction of properties was the REA-DSL's goal to not only provide a modeling tool for business models but also to semi-automatically create an Entity-Relationship-Diagram (ERD) for an AIS [41]. Giving the user the opportunity to define the aforementioned properties enables the specification of columns for the tables and their datatypes.

Agent View

The agents participating in the activities of the car producer value chain have to be defined as well. Using the REA-DSL, these agents are modeled as stick figures where white-headed figures are inside agents and black-headed figures are outside agents. The following agents are defined: *employee*, *salesperson*, *shop assistant*, *cashier*, *clerk*, *manufacturer*, *factory*, and *customer* (cf. Figure 4.2).

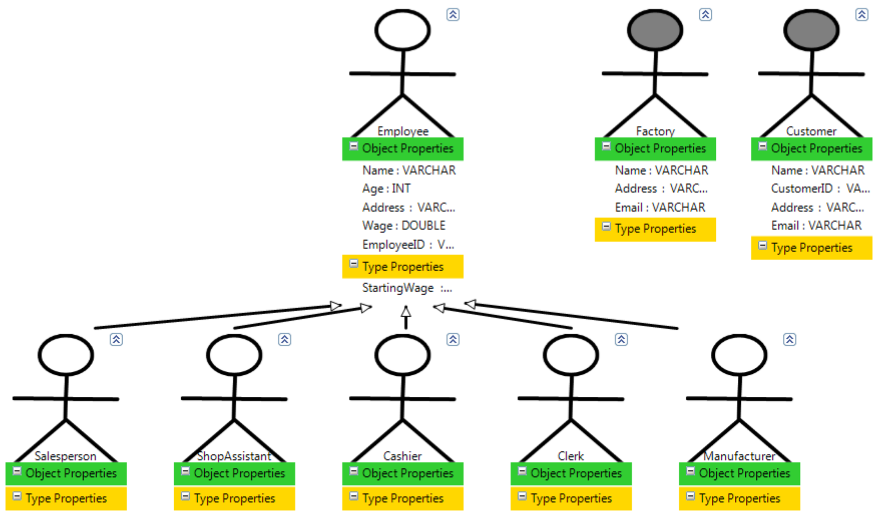


Figure 4.2: REA-DSL Agents

Agents can have object- (green compartments) and type-properties (yellow compartments) like the aforementioned resources. `Salespersons`, `shopassistants`, `cashiers`, `clerks`, and `manufacturers` are specializations of `employees` and therefore inherit their properties (`name`, `age`, `address`, `wage`, and `employeeID`). Furthermore, each of these five employee subtypes defines a minimum `startingWage`. Such type properties on agents can define properties for policies that must be fulfilled. For instance, the `wage` object property of a `cashier` must not be lower than his `startingWage` type property that specifies a cashiers starting wage.

`Factory` and `customer` are outside agents. The car producer buys the `raw material` from the `factory` producing it. `Customers` are buying the produced `cars`. Both include a `name`, an `address`, and an `email` object property. `Customers` additionally possess a `customerID`.

Value Chain View

Agent view and resource view form the foundation for the other three views. The already defined agents, agenttypes, resources, and bulk resources can therefore be used in the valuechain-, duality-, and planning view. When modeling the value chain of the car producer only the resources are used as can be seen in Figure 4.3.

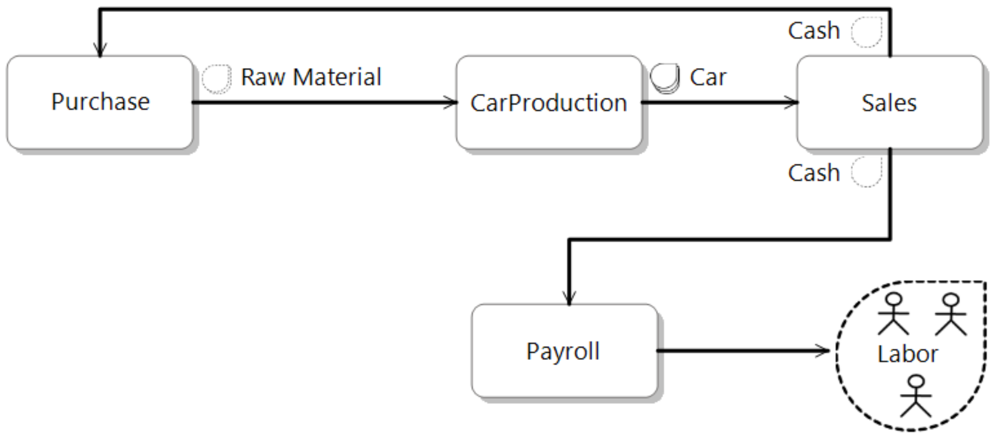


Figure 4.3: REA-DSL Value Chain

The value chain modeled with the REA-DSL looks similar to the sketched value chain in Figure 3.5 on page 31. The four business activities are modeled as rounded rectangles. Each one of them can either be identified as transformation (conversion) or transfer (exchange) activity.

Resources flowing between the business activities are depicted as drops located on top of the arrows pointing from one activity to the succeeding one. The direction of the arrow defines the resource flow. These resources were already created in the resource view and are only referenced now. Besides the solid or dashed drops that stand for resources and bulk resources, also stacked drops (so called resourceseries or resourcetypeseries) can occur. An example is the stacked resource `car` in Figure 4.3 flowing from `car production` to `sales`. During the `car production` activity not only one `car` is produced. Instead multiple resources of the same kind can be manufactured. The same happens in the `sales` activity where not only one but multiple `cars` are sold.

A difference to the resource flow in Figure 3.5 on page 31 is the usage of `labor`. In contrast to using it as input for the `car production` activity it is only defined as outcoming resource of the `payroll` activity. The reason for this is that `labor` is not consumed or used during a stock-flow relationship with a decrement event. Instead it is used when an inside agent participates in an action which happens in nearly every event. Therefore, it is not explicitly modeled [41].

Operational View and Planning View

Each of the four business activities, depicted as rounded rectangles (cf. Figure 4.3), can be modeled in more detail as duality between events or reciprocity between commitments. It is possible to automatically create the events in the operational view or commitments in the planning view as stubs. Since in our example there is no difference in the concepts of the planning- and operational view, we just model the planning view and derive the operational view from it. Events that will actually occur in the future are planned by commitments that are defined for eventtypes. An example is a contract to buy `raw material` during the `sales` activity or a schedule to produce `cars` in the `car production` activity. Therefore only planning views will be created for each of the four activities by double clicking on them in the value chain view. A planning view stub is created that consists of one increment and one decrement eventtype and resources/resourcetypes reserved by them. Again resourceseries and resourcetypeseries are also possible. For conversion activities one inside agent will be automatically created for each eventtype. Exchange activities have one inside agent and one outside agent attached after the creation. Only minor changes are needed for users of the REA-DSL in order to adapt the created models to their expectations and needs by changing the names of participating agents or adding more agents.

Figure 4.4 shows the planned REA-DSL sales activity that was illustrated in class-diagram style in Chapter 3.5. Already at the first glance it becomes obvious that the class-diagram-like REA format is much more complex (cf. Figure 3.7 on page 35). In REA-DSL models `commitments` are depicted as scrolls. Each commitment has one agent that legally commits to it. Increment commitments (2) are always in reciprocity with decrement commitments (1) and vice versa. During the sales activity plan, the agent `customer` commits to pay the bulk resource `cash` to the `cashier` during the `cash receipt` event. In contrast the `salesperson` commits to provide the sold `cars` to the `customer` during the `sale` event. `Shopassistants` are also participating in that event. Since not only one but several assistants (who are not known at the moment) can be involved, they are modeled as stack of agenttypes which depicts them as agenttypeseries. The planning view only shows dashed hexagons and therefore eventtypes (3,4). The actual events are modeled in the operational views of the activities. Properties can also be defined for the eventtypes. Event properties in green are set for each individual sale when it occurs (`saleNr`, `saleDate`, `receiptNr`, `receiptDate`). Event type properties in yellow define properties for a special kind of event (`region` defining in what region an event with a specific `saleNr` happened). Additionally commitment properties can be defined in purple that describe characteristics of what is planned or scheduled. Figure 4.4 has two eventtype commitmentproperties: `orderNr` and `orderDate`. Both define properties of the selling contract between car producer and the customer. The agents were already defined in the agent view and are just referenced in planning view. The same applies to resources. The Stockflow-, policy-, and reserve properties that are defined below resources (5) represent properties for the stock-flow relationships. For example the `actualPrice` of the sold cars is the amount of money received during the `sale` event. Type properties defined for stock-flow relationships can be used to define policies. It therefore is possible to specify the `standardPrice` for products sold during types of events (a car sold during a sale occurring in the austrian region). Commitment properties can be used to define the price that was set in the contract between customer and seller (`committedPrice`). Properties below agents (6) are defined for participate relationships. The participate property `customerSatisfaction` therefore describes how satisfied the customer was with the salesperson during the sales event and has to be set for each event individually. The

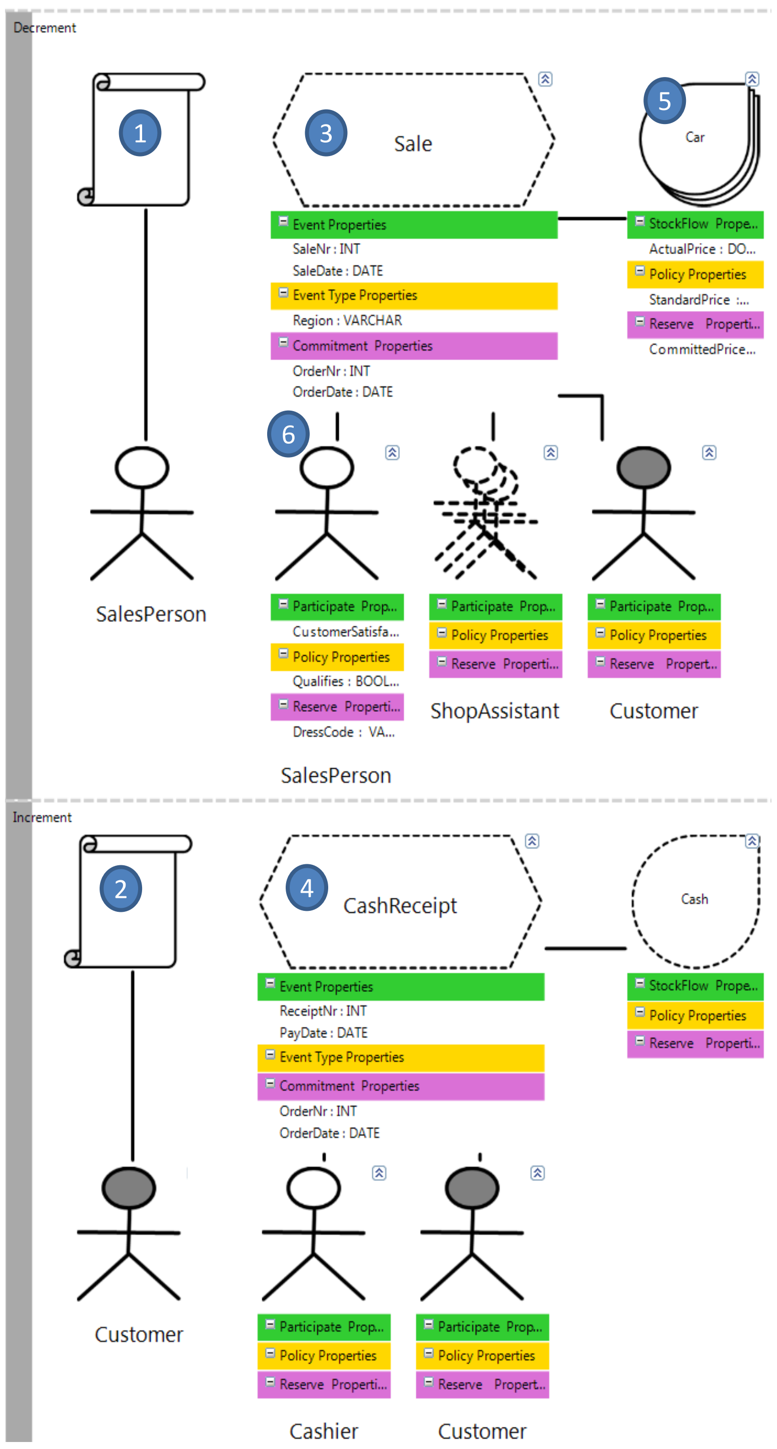


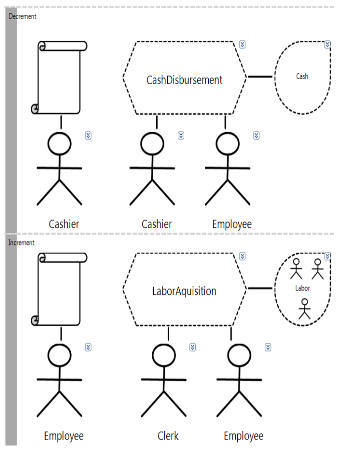
Figure 4.4: REA-DSL Planned Sales Activity

policy property `qualifies` is a policy describing which agents should be able to participate in what type of events. `DressCode` is a reserve property describing characteristics of the contract and can be used to define what sellers should wear when selling cars to the customer. The money gained from the sales activity is used to pay employees (payroll activity) and to purchase raw material (purchase activity). Again planning views are created to show these activities in a more detailed way. Figure 4.5 shows the remaining three planned activities with collapsed properties.

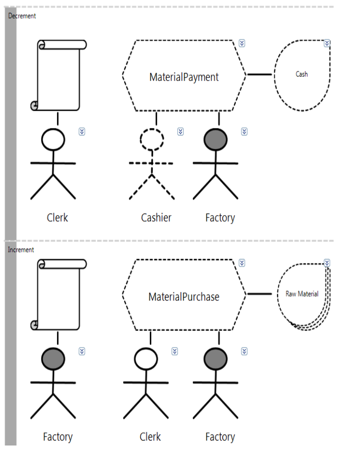
In the planned payroll activity (cf. Figure 4.5a) a `cashier` commits to pay `cash` to an `employee` during a `cashdisbursement` event. In exchange the `employee` commits to provide `labor` to the `clerk`.

Figure 4.5b shows the planned activity where raw material is purchased for money. A `clerk` commits to the contract that a `cashier` (the concrete cashier that will do the payment is not known. Therefore an agenttype is used to indicate that every cashier can participate in the event) will pay `cash` to the raw material producing `factory` during the `materialpayment` event. The factory on the other hand commits to deliver the `raw material` (which can be `steel`, `plastic`, `aluminium`, `rubber`, or `glass` as depicted in Figure 4.2 on page 41) to the `clerk`[2].

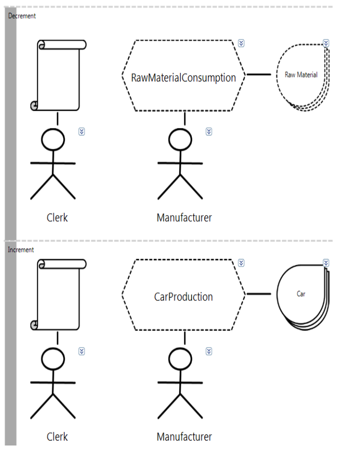
Finally in the car production activity (cf. Figure 4.5c) a `clerk` commits to the scheduled car production where `manufacturers` consume `raw material` to produce `cars`.



(a) Planned Payroll Activity



(b) Planned Purchase Activity



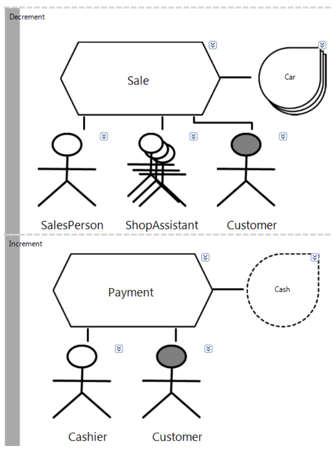
(c) Planned CarProduction Activity

Figure 4.5: REA-DSL Planning Views of the Payroll, Purchase, and Car Production Activity

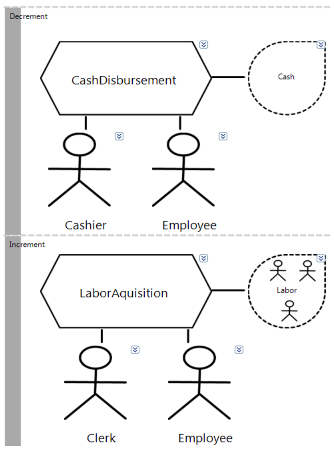
In the four aforementioned activities no properties are shown. Nevertheless these properties do exist. When creating business models using the REA-DSL the properties can be hidden to guarantee better readability of the model. Clicking on the small arrow on the top right of agents, resources, events, commitments, participate relationships, or stock-flows enables the user to show or hide the additional information.

[2]In this simple example it is assumed that one factory produces all the raw material necessary to produce a car

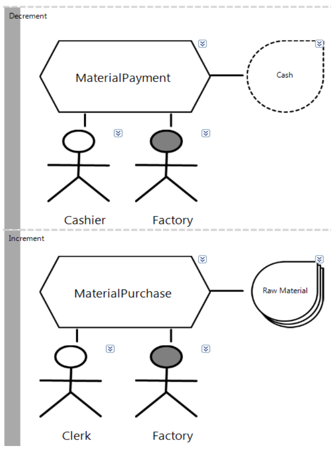
The operational views of the four activities (cf. Figure 4.4 and Figure 4.5) can easily be derived from the planning layer by substituting eventtypes/eventtypeseries by events/eventseries and removing commitments from the model. Figure 4.6 on page 46 shows these views for the sales (4.6a), payroll (4.6b), purchase (4.6c), and the car production activity (4.6d).



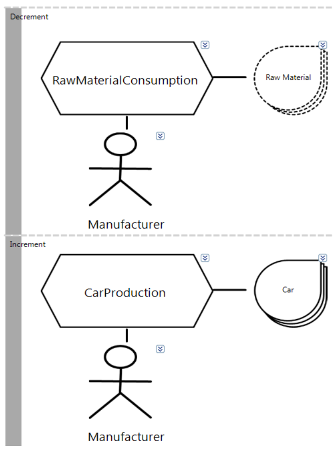
(a) Sales Activity



(b) Payroll Activity



(c) Purchase Activity



(d) Carproduction Activity

Figure 4.6: REA-DSL Operational Views of the Carproducer Activities

As illustrated by the example company of the car producer, using the REA-DSL to create business models is quite simple. The models are based on REA concepts and include resources occurring in the activities of a company, events affecting them, agents participating in the events, and commitments that describe what is planned or scheduled. Compared to the class-diagram-like representation of McCarthy, the concepts are visualized in an intuitive way which makes them more understandable. Furthermore, the created business models can be adapted easily.

4.3 Using REA-DSL Models to create a Data Structure for AIS

As mentioned in Chapter 4.2 the REA-DSL can also be used to create an Entity-Relationship-Diagram (and therefore the relational data structure) of an AIS. Business experts and IT-professionals can use the DSL together and define the conceptual model of the system. Instead of remodeling REA-DSL constructs to a relational model that adheres to the rules of the REA ontology, these model can automatically be created based on a mapping proposed by Mayrhofer [41]. The advantage of this approach is that errors, that might occur during the remodeling of the constructs by the IT-professional, are avoided. Furthermore, time is saved since these steps are carried out by the DSL itself.

The REA-DSL concepts, needed to make a mapping to the relational model possible, were already identified in the previous chapter as object-, type-, event-, event type-, commitment-, stockflow, participate, and reserve properties. Each of these concepts is specified by a name and a type. Additionally they can be marked as primary key which identifies a property as unique identifier for a REA-entity [42]. Since users of the REA-DSL are able to define primary keys and properties, all the information that is necessary to describe the relational database schema of an AIS is contained [41]. The mapping is done by the *Microsoft Visual Studio T4 Text Templating Engine* [7]. These T4 text templates are used to create SQL files for the table creation and just need to be executed by a database management system.

CHAPTER 5

The REAlist Project and REA DB

5.1 Scope of the REAlist Project

As explained in the previous chapters, ERP systems support enterprises to execute their business activities. These activities might look similarly but the concrete realizations differ from business to business. Therefore a business standard software cannot be used immediately after the installation but has to be adapted to the specific needs of the company that uses the system. The major drawbacks of current ERP systems (complicated customization leading to high follow-up costs and missing business semantic base) already were extensively discussed in Chapter 2. In order to overcome these problems the REAlist project was proposed by the Business Informatics Group (BIG)[1] at the Vienna University of Technology in [34]. The Eventus Marketingservice Gmbh[2] acts as business partner.

The goal of the REAlist project is to provide a cost-saving approach for company-specific ERP system adaptations and customizations. Compared to existing solutions, where separate instances of the software are provided to every customer, the system is hosted as single instance in the cloud (cf. Figure 1.1 on page 2). Multitenancy is a crucial point in the project since all customers use the same instance of the application and therefore the same database.

The main data structure of the underlying database (REA DB) is generic. This means that even if current business needs change or new requirements occur, they can be saved without changing the structure. As mentioned in Chapter 3, business needs can be represented as business models. One way to illustrate such business models is the REA ontology that encompasses all concepts that are relevant in ERP-Systems, like types, policies, and commitments. The REA DB is based on this ontology and therefore capable of saving all kinds of REA business models. Actual business data is stored in the generic part of the database. This overcomes the aforementioned drawback of existing ERP systems, where the data structure needs to be adapted when the market demands change. As long as the adapted business models are based on REA concepts as well,

[1]http://www.big.tuwien.ac.at/
[2]http://www.eventus.at/

they can be saved to the REA DB. The feasibility of that REA-based database for ERP-Systems is shown in [44] and will be discussed extensively in Chapter 5.2. The medium-term goal of the REAlist project is to implement a prototype hosted on servers of cloud providers that can be accessed by users over a website using an internet browser.

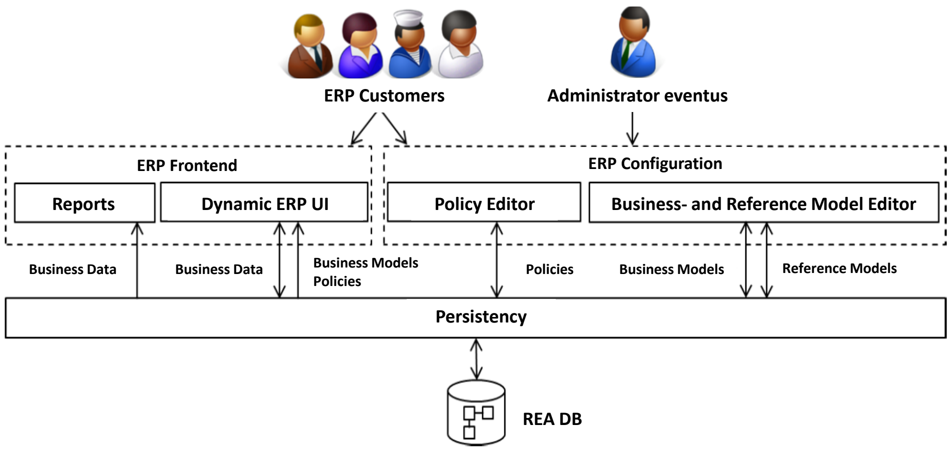


Figure 5.1: Functional Scope of the REAlist Project [34]

The functional scope of the REAlist project is illustrated in Figure 5.1. In order to enable the company-specific customizations of the application, a model-driven approach is used. Business needs, that might occur, are represented as REA `business models`. `Customers` (tenants) or `administrators` of the ERP-System describe their models using the `business- and reference model editor`. As modeling language the REA-DSL (cf. Chapter 4) is proposed since it enables users to create `business models` in a clear and simple way 4. Furthermore, `reference models` can be defined and used as foundation for other models. Doing so saves time since modelers do not have to start from scratch when formulating their business needs. The `business models` and `reference models` are persisted in the REA DB. Since the database and the business models are both based on the REA ontology, a mapping of the models to the database is possible. REA business operations can be restricted by `policies`. Therefore a user-friendly way of applying such rules to the models, without adapting the code or the data structure, is proposed. Policies specify guidelines or constraints and declare what could, should, or must happen as already mentioned in Chapter 3.5. Geerts et. al [18] proposed three different policy types: *knowledge intensive descriptions* describing characteristics of objects, *validation rules* specifying thresholds for such characteristics, and *target descriptions* describing planned objectives. The sales activity of the car producer that was illustrated in the previous chapter (cf. Figure 4.4 on page 44) defined the region for sale events as knowledge intensive description. The standard price of cars is a target description of the actual price specifying what should be. A validation rule is illustrated in Figure 4.2, where the minimum wage for employees is defined as type property. Another example for policies are sales discounts in a sale event for new customers exceeding a certain bill amount. Using the `policy editor` policies are defined by `customers` or the `administrator` with a policy language that references REA objects and automatically checks the saved models for correctness. The policies (and therefore the business case rules) are also persisted in the REA DB. Based on the persisted `business models` and `policies`, user interfaces should be generated automatically for the `customers` as `dynamic ERP UI`. The interface has to reflect all the relevant parts that were specified in the REA business models. For example a saved model consisting of resources, agents, and dualities should result in the generation of masks where these entities can be created, updated, or deleted. When creating a duality of a specific type, events, stockflows, and participations are appended in the interface according to the definitions in the underlying business model. The same happens for additional attributes that are added for entities. `Customers` can use the generated user interfaces to conduct business cases. The entered `business data` is saved to the generic part of the database and can be checked against the business models for validity. Furthermore, meaningful reports and statistics should be created from the `business data` in the REA DB. Due to the REA based data structure it is possible to completely reproduce which payments where transferred during which events [34]. The project name REAlist already indicates the possibility to create balance **list**s based on a **REA** database.

5.2 The REA Database (REA DB)

The core of REAlist is the data structure based on REA concepts. The REA DB is divided in two parts as shown in Figure 5.2. The business model part includes all tables that are necessary to store the REA-DSL business models (1). The tables that are important for saving the actual business cases (2) that are conducted by users are comprised in the generic part of the database. Since it may store any kind of business data, this part is called generic. The validity of that data can be checked against the business model data. During a feasibility study Mayrhofer et. al [44] proved that such an approach can be realized. Several business cases were investigated and represented with the REA data structure.

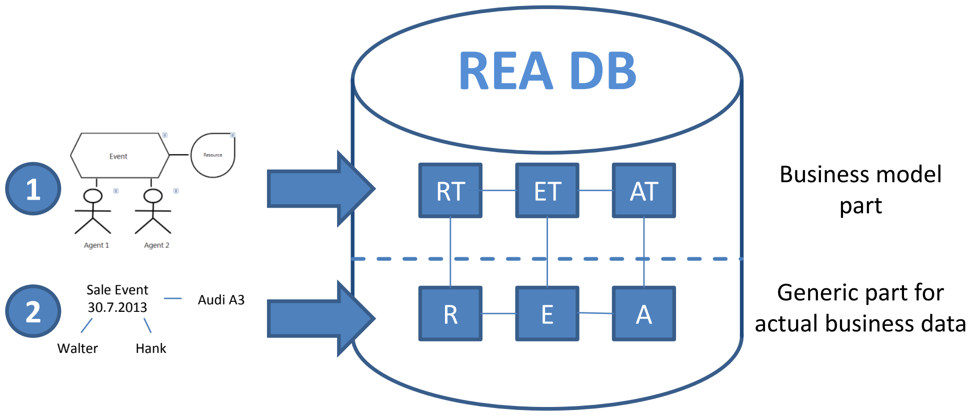


Figure 5.2: Parts of the REA DB

The concept of the generic database is illustrated in Figure 5.3. Layer M2 represents the core elements of the ERP data structure. These elements encompass all the concepts of the

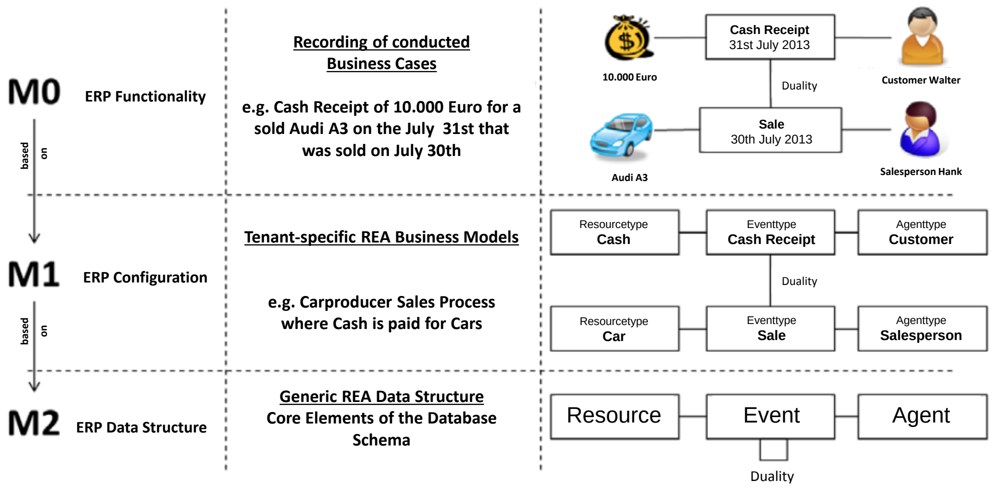

M0
ERP Functionality
Recording of conducted Business Cases
e.g. Cash Receipt of 10.000 Euro for a sold Audi A3 on the July 31st that was sold on July 30th
10.000 Euro
Cash Receipt 31st July 2013
Customer Walter
Duality
Audi A3
Sale 30th July 2013
Salesperson Hank
based on
M1
ERP Configuration
Tenant-specific REA Business Models
e.g. Carproducer Sales Process where Cash is paid for Cars
Resourcetype Cash
Eventtype Cash Receipt
Agenttype Customer
Duality
Resourcetype Car
Eventtype Sale
Agenttype Salesperson
based on
M2
ERP Data Structure
Generic REA Data Structure Core Elements of the Database Schema
Resource
Event
Agent
Duality

Figure 5.3: Layers of the REA based Data Structure [44] (adapted)

REA ontology and therefore resources, events, agents, commitments, policies, types, and the relationships between those entities. Therefore the economic principles of REA are bound to the data structure permitting REA based business models and concrete business cases to be saved without changing it. On M1 business- and reference models are stored that define how concrete business cases are structured. In Figure 5.3 the simplified sales activity of the car producer is illustrated (for a more detailed class-diagram-like REA representation of that activity cf. Figure 3.4 on page 30). From the car producers point of view, `cars` are sold by the `salesperson` to the `customer` during the `sale` event. In exchange the `customer` pays `cash` during the `cash receipt` event to the `salesperson`. The two events are in a `duality` relationship (in this case an exchange-duality) since they are compensating each other. In these models type entities of resources, events, and agents are used. M0 records the business cases that are conducted. For the sales activity example a business case could look like the following. The customer `Walter` decides to buy a car from the car producer. `Hank`, the salesperson, therefore sells an `Audi A3` to `Walter` during a `sale` on July 30th 2013. `Walter` pays `10.000 Euros` for the acquired product to `Hank` on the following day.

Generic Part of the REA DB

The data structure of the REA DB consists of various entities, structured in ten subparts: *REA-Core*, *Resources*, *Location*, *REA-Constellations*, *Status*, *Negotiation*, *Policy*, *Currency and Country*, *AdditionalAttributes* ,and *Price List* [44]. Not all of these parts will be explained in detail. Instead the most relevant ones are discussed and a brief overview of the remaining ones will be given. The REA DB has been modeled using MySQL Workbench[3]. REA-Core consist of two parts: one consisting of the most important REA ontology entities (resources, events, and agents) and one extending them by the duality concept, commitments, contracts, and claims.

[3]`http://www.mysql.com/products/workbench/`

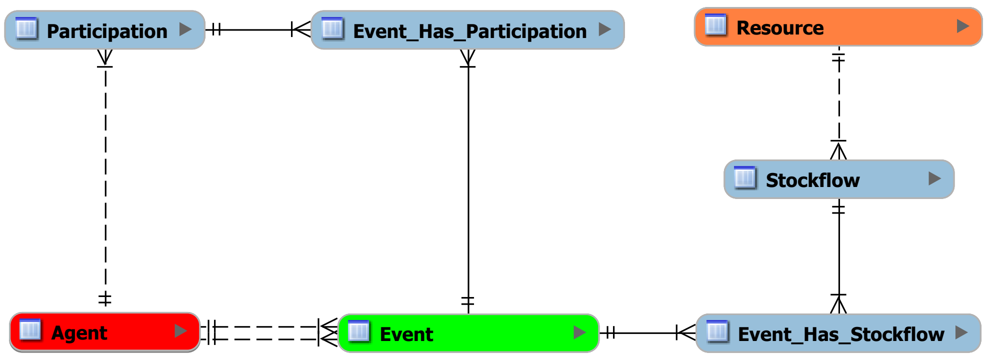


Figure 5.4: Core Elements of the generic REA DB

The core elements of the REA based data structure are shown in Figure 5.4 as entity-relationship diagram. Resources, events, and agents are represented in the **Resource**, **Event**, and **Agent** tables. Events have one agent that provides and one that receives resources. Furthermore, several agents can participate in an event. This information is stored in the **Participation** and **Event_Has_Participation** tables. The relationship between resources and the events, during which they are affected, is represented in the **Stockflow** and **Event_Has_Stockflow** tables.

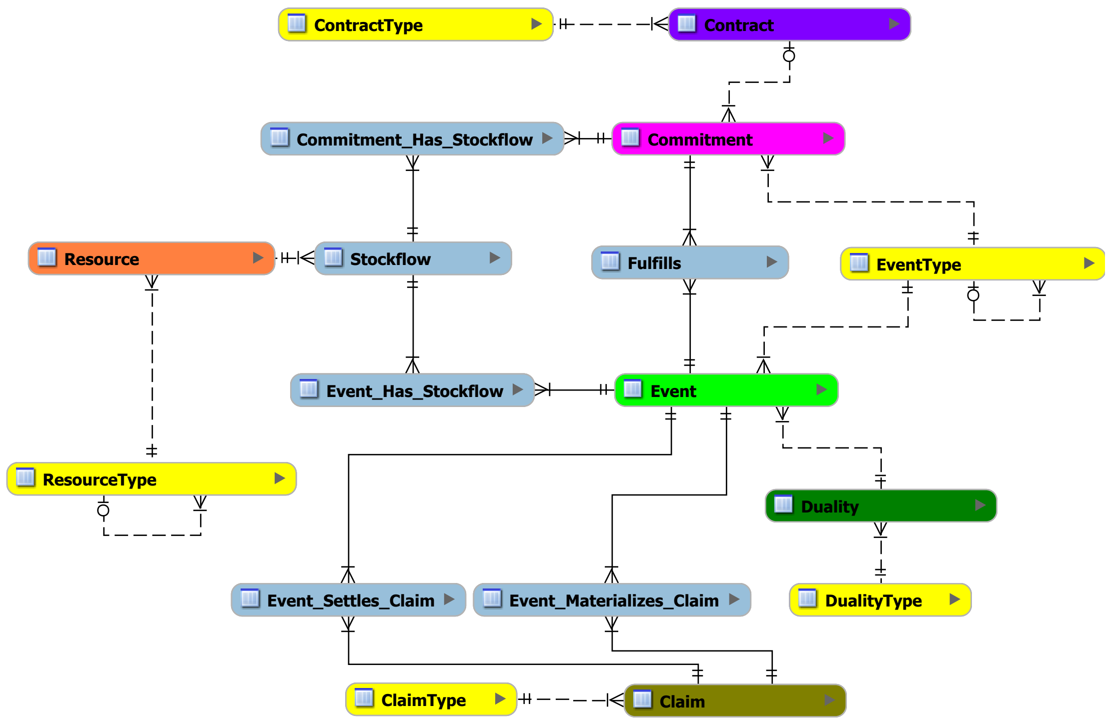


Figure 5.5: Extended Core Elements of the REA DB

Figure 5.5 illustrates the extensions to these core elements. The ERD includes a **Duality** table that combines events to a business case. Events can be preceded by a commitment (represented in the **Commitment** table). Entries of the **Fulfills** table relate commitments to the corresponding events. Both entities are related to an **Eventtype**. The fact that resources are reserved by a commitment is represented in the **Commitment_Has_Stockflow** table. A **Claim** describes which event has not yet been compensated by an opposing event. Type entities for contracts, resources, claims, and dualities are represented in the **ContractType**, **ResourceType**, **ClaimType**, and **DualityType** tables. It has to be noted that entities for agents and their corresponding types are not visualized in the ERD. Furthermore, the participation relationships between events/commitments and the agents are omitted. The tables relevant for these concepts are **Agent**, **Agenttype**, **Participation**, **Event_Has_Participation**, and **Commitment_Has_Participation**. In Figure 5.4 and 5.5 the attributes are hidden to ensure better readability.

The remaining parts of the generic REA DB extend the REA core concepts by all sorts of information that is needed in ERP-Systems. The corresponding ERD's are listed in Appendix A. Details of resources are covered in the Resources part. Besides the already illustrated **Resource** and **Stockflow** tables, **IdentifiableResource** is used to describe resources that are affected during stockflows in greater detail. E.g., 100 iPads are exchanged during a stockflow. Each of these iPads is represented as entry in **IdentifiableResource** with its appropriate `serialnumber`. Taxes or discounts can be defined in the **Tax** and **DiscountOrAddition** table that relate resources with stockflows. **Resource_Composed_Of** allows to specify that a resource consists of several resources itself. It therefore defines the linkage relationships that where described in Chapter 3. In order to persist addresses in the REA DB, the Location subpart is used. A **Location** can either be represented as entry in the **Location_Address** table, in form of a `city`, `street`, and `zip code`, or as entry in the **Location_Coordinates** table, with `X` and `Y` coordinates. The **LocationType** table is used to additionally categorize addresses (e.g., as shipment address). Locations can be declared for all REA concepts (e.g., resources, events, agents, identifiableresources, commitments). Since currencies are important for accounting, the Currency and Country subset of the REA DB is dedicated to this concept. Entries of the **Currency** table are related to **Country** table entries. Among others, currencies can be defined for events, commitments, claims, and stockflows. In the Price List subpart, price lists can be defined for specific locations, agenttypes, and time periods in the **PriceList** table. **PricePolicy** depicts the price of resources in a pricelist. The Negotiation part is used to keep track of the negotiation status until a contract is fulfilled. A **Negotiation** can represent the demand and supply cycle in form of proposals. A **Proposal** defines who is proposing it (`proposingAgent_Id`) and if the offer is binding or not (`isBinding` attribute). `ValidDueDate` specifies how long the offer is valid. Preceding proposals can be specified with the `ParentProposal_Id` attribute. Proposals can include several potential commitments that do not need to be entirely specified while the negotiation is still ongoing. After a negotiation is finished, commitments are transferred to a **Contract**. To enable a reconstruction of the contracts negotiation process, the **Contract_Has_Negotiation** table relates contracts to negotiations. Information about the status of negotiations, contracts, dualities, or claims is covered by the Status subpart. The REA DB also enables the saving of policies. Therefore the Policy part can be used to define possible formations of resource-, event-, and agenttypes. As argued in [44] only very simple policies are possible. Since one goal of the

REAlist project is to define a new policy language that can also be saved in the REA DB, this subsection might only exist temporarily.

Business Model Part of the REA DB

The concepts of the business model part where the REA-DSL models will be saved are REA-Constellations and AdditionalAttributes. REA-Constellations defines the formations of the REA core concepts on a higher level. It therefore is possible to specify how an actual business case, that will occur in the future, is structured. Tables for contract-, duality-, event-, resource-, agenttypes, and the relationships between these concepts are included. AdditionalAttributes encompasses tables for the attributes that can be defined for resources, events, agents, stockflow-, and participation relationships. As explained in Chapter 4, these attributes are represented in a REA-DSL model in form of properties. The possible manifestations of properties that can be specified on entities were depicted as object-, type-, commitment-, event-, event type-, participate-, stockflow-, policy- and reserve properties.

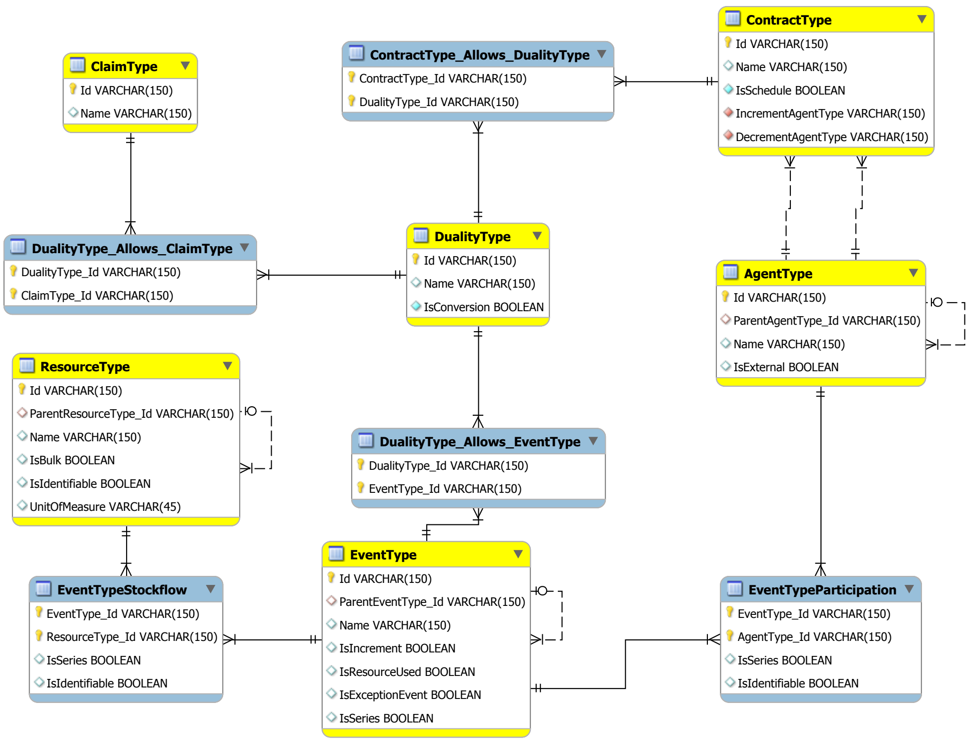


Figure 5.6: REA Constellations

Figure 5.6 shows the entity-relationship diagram of the REA-Constellations including tables and attributes. Since REA-DSL models only specify a blueprint for actual business cases that will be conducted later on, elements of the models (e.g., events, affected resources, and participating agents) are saved in type tables. The join tables between the REA types relate these elements which allows us to save REA-DSL models in the REA DB.

All declared resources are saved to the **ResourceType** table including a `Name` and, since a resource hierarchy can exist, a `ParentResourceType_Id` attribute. `IsBulk` and `IsIdentifiable` are used to differ between bulk resources, and identifiable resources.

AgentType is used to store the declared agents that can have superagents and a name (`ParentAgentType_Id` and `Name`). Inside and outside agents can be distinguished with the `IsExternal` attribute.

DualityType specifies the duality relationship that is defined in the REA-DSL model. The fact that a duality can either be a conversion or an exchange is expressed in the `IsConversion` attribute. Furthermore, a `Name` attribute specifies how the duality is named.

ContractType stores the contracts that also have a `Name` and can either be a contract or a schedule (depicted by the `IsSchedule` attribute). Two agents are responsible that the contract is fulfilled. Therefore one `IncrementAgentType` and one `DecrementAgentType` attribute is included that references the corresponding agenttypes.

The **EventType** table is capable of storing the events that have been defined in the REA-DSL's operational- and planning view. Besides a `Name` and `ParentEventType_Id` attribute, `IsIncrement` differs between increment and decrement events. `IsResourceUsed` depicts if resources are used or consumed during the event, and `IsSeries` defines if the event/eventtype is a series in the model or not.

The join tables (blue) in Figure 5.6 represent relationships between REA concepts. For example **DualityType_Allows_EventType** defines what eventtypes occur in a specific duality. In most of these tables only the IDs of the two connected tables are specified. Notable exceptions are **EventTypeStockFlow** and **EventTypeParticipation**. Both include an `IsIdentifiable` attribute since resources and agents can also occur as types in the REA-DSL operational- and planning views. Additionally a `IsSeries` attribute exists for agent- or resource series in the REA-DSL model.

Figure 5.7 shows the ERD including the additional tables of the REA DB, that are needed to store all kinds of properties. EventTypeStockflow, EventTypeParticipation, Resource-, Event-, and AgentType tables were already explained, but since these are the concepts that can have properties, they are also visualized in the AdditionalAttributes.

Every property that has been defined in one of the REA-DSL views is stored in the **Attribute** table with its defined `Name` and `DataType`.

The binding of these properties to the corresponding model elements is saved in the join tables (blue) of Figure 5.7. Event- or Eventtype properties are saved in the **EventType_-Has_AdditionalAttribute** table, where `IsTypeProperty` and `IsCommitmentProperty` differ between type- and commitment properties. If each of the these attributes is set to false, a property is identified as simple event property. **EventTypeStockFlow_Has_AdditionalAttribute** and **EventTypeParticipation_Has_AdditionalAttribute** include similar attributes (`IsPolicyProperty`, `IsReserveProperty`) and are used to store properties of stockflow- and participation relationships. **ResourceType_Has_AdditionalAttribute** and **AgentType_Has_AdditionalAttribute** only encompass an attribute to identify type properties (`IsBulkProperty`, `IsTypeProperty`) since commitment properties can not be defined in the agent- and resource view of the REA-DSL. `IsOptional` exists in each of these tables to identify, if properties need to be defined or not.

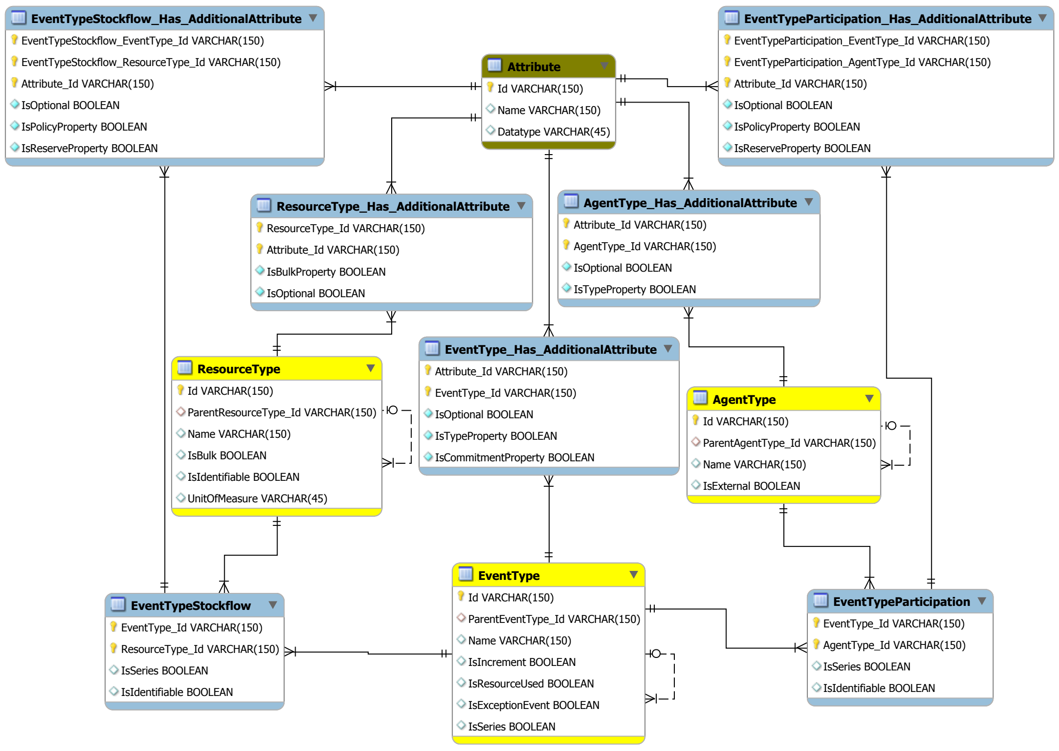

EventTypeStockflow_Has_AdditionalAttribute
EventTypeStockflow_EventType_Id VARCHAR(150)
EventTypeStockflow_ResourceType_Id VARCHAR(150)
Attribute_Id VARCHAR(150)
IsOptional BOOLEAN
IsPolicyProperty BOOLEAN
IsReserveProperty BOOLEAN
Attribute
Id VARCHAR(150)
Name VARCHAR(150)
Datatype VARCHAR(45)
EventTypeParticipation_Has_AdditionalAttribute
EventTypeParticipation_EventType_Id VARCHAR(150)
EventTypeParticipation_AgentType_Id VARCHAR(150)
Attribute_Id VARCHAR(150)
IsOptional BOOLEAN
IsPolicyProperty BOOLEAN
IsReserveProperty BOOLEAN
ResourceType_Has_AdditionalAttribute
ResourceType_Id VARCHAR(150)
Attribute_Id VARCHAR(150)
IsBulkProperty BOOLEAN
IsOptional BOOLEAN
AgentType_Has_AdditionalAttribute
Attribute_Id VARCHAR(150)
AgentType_Id VARCHAR(150)
IsOptional BOOLEAN
IsTypeProperty BOOLEAN
ResourceType
Id VARCHAR(150)
ParentResourceType_Id VARCHAR(150)
Name VARCHAR(150)
IsBulk BOOLEAN
IsIdentifiable BOOLEAN
UnitOfMeasure VARCHAR(45)
EventType_Has_AdditionalAttribute
Attribute_Id VARCHAR(150)
EventType_Id VARCHAR(150)
IsOptional BOOLEAN
IsTypeProperty BOOLEAN
IsCommitmentProperty BOOLEAN
AgentType
Id VARCHAR(150)
ParentAgentType_Id VARCHAR(150)
Name VARCHAR(150)
IsExternal BOOLEAN
EventTypeStockflow
EventType_Id VARCHAR(150)
ResourceType_Id VARCHAR(150)
IsSeries BOOLEAN
IsIdentifiable BOOLEAN
EventType
Id VARCHAR(150)
ParentEventType_Id VARCHAR(150)
Name VARCHAR(150)
IsIncrement BOOLEAN
IsResourceUsed BOOLEAN
IsExceptionEvent BOOLEAN
IsSeries BOOLEAN
EventTypeParticipation
EventType_Id VARCHAR(150)
AgentType_Id VARCHAR(150)
IsSeries BOOLEAN
IsIdentifiable BOOLEAN

Figure 5.7: REA AdditionalAttributes

The mapping from the defined REA-DSL models to the generic database is described in the next chapter. Chapter 7 deals with the creation of the user interface and the storage of the conducted business cases in the REA DB. In this section, REA Core and the relevant subset of AdditionalAttributes that is needed to persist business cases, will be investigated in greater detail.

CHAPTER 6

Mapping REA-DSL Models to the REA DB

6.1 Considerations for the Mapping Process

The core of the REAlist project is the REA DB that consists of the business model part for storing REA-DSL models and the generic part capable of saving actual business cases. This database is based on REA concepts and was introduced in the previous chapter. The data structure of the REA DB includes a lot of the concepts relevant for ERP systems and is capable of storing economic phenomena that are specified using REA concepts. A system based on that structure therefore does not need to be adapted drastically in order to fulfill new business needs. Since these needs are represented as REA-DSL models, a mapping from these models to the REA DB has to be defined. The parts of the database that are used to store the defined business models were already identified and explained in Chapter 5.2.

In order to save REA-DSL models to the REA DB, insert statements have to be generated for the database. These statements have to encompass all entities that were defined in the respective business models. The database management system of choice for the purposes of this thesis is *MySQL*[1] since the REA DB database schema was modeled with *MYSQL Workbench*[2]. The database can easily be created based on this schema and manipulations like populating the tables can be achieved by using plain SQL. The mapping between the REA-DSL models and the REA DB tables will be discussed next. Based on the defined models of the car producer in Chapter 4 mapping rules will be defined for the resources, agents, and the activities that were specified.

[1]http://www.mysql.com/

[2]http://www.mysql.com/products/workbench/

6.2 Mapping Resources to the REA DB

Saving REA-DSL resources to the REA DB demands that all elements are somehow mapped to one of its tables. A model containing resources that are affected in the value chain of an enterprise was already presented in Chapter 4 (cf. Figure 4.1 on page 40). This model contains, apart from unique identifiable resources, all possible manifestations of resources that can be defined using the REA-DSL: bulk resources (cash, raw material, and its subtypes), identifiable resources (car and subtypes), and the labor resource (labor). Therefore parts of this model will be used to explain the intended mapping. The tables that are relevant to store the aforementioned resources and their properties are **ResourceType**, **Attribute**, and **ResourceType_Has_AdditionalAttribute**.

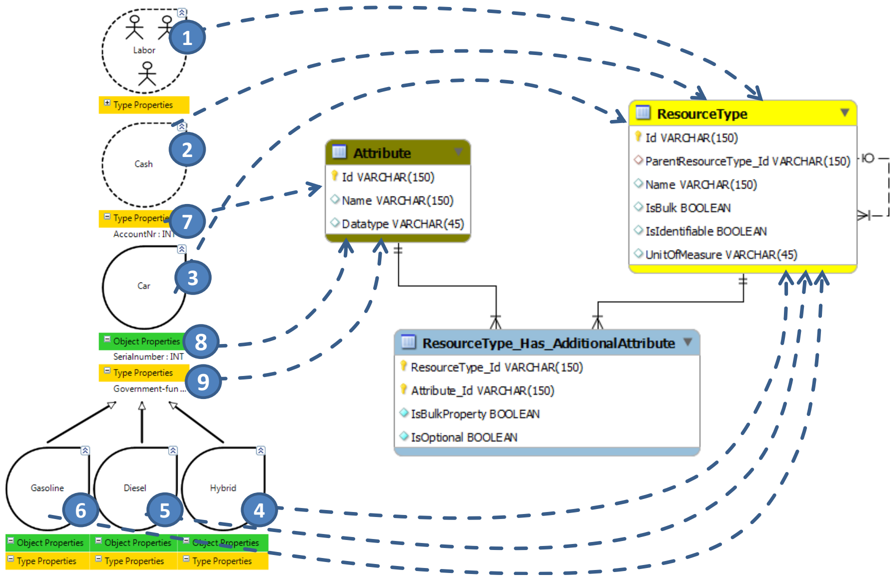


Figure 6.1: Resource Mapping

Figure 6.1 shows how REA-DSL resources are mapped to the REA DB. Algorithm 6.1 on page 61 describes this procedure in greater detail. All kinds of resources are represented as entries of the **ResourceType** table. For labor (1) and cash (2) the `IsBulk` value has to be true. The defined name and the unit for the quantity on hand are represented as `Name` and `UnitOfMeasure` values in the table. Car (3), hybrid (4), diesel (5), and gasoline (6) are identifiable resources since they have object- and type properties. Therefore they are distinguished from bulk resources by only setting `IsIdentifiable` to true. Their name is represented as `Name` value in the table entry. Hybrid (4), diesel (5), and gasoline (6) are subtypes of car. This fact is indicated by the `ParentResourceType_Id`. The three aforementioned resources therefore define a foreign key that identifies their supertype. The mapping of unique resources is not shown in Figure 6.1, but such resources are also mapped to the **ResourceType** table with `IsBulk` and `IsIdentifiable` values set to false. Type- and object properties are represented in the **Attribute** table with their `Name` and `Datatype`. The relation between resources and their properties is stored in the **ResourceType_Has_AdditionalAttribute** table. In case of type properties like the account number (7) and government-funded (9), `IsBulkProperty` is true. For object properties like serialNumber (8), `IsBulkProperty` is set to false. `IsOptional` is defined as false for all attributes of resourcetypes since the REA-DSL currently does not provide the possibility to declare properties as optional.

input : REA-DSL resource model $rdsl\_rm$
output : Updated REA DB after inserting resourcetype entries re, attribute entries ae, and resourcetype_has_additionalattribute entries $rhaa$

```
foreach Resource sr in rdsl_rm do
    // setting attribute values for resourcetype entries
    re.Name ← <sr.Name>;
    re.unitOfMeasure ← null;
    re.isBulk ← false;
    re.isIdentifiable ← false;
    re.parentResource ← <sr.Super>;
    if sr is a ResourceType then
        re.isBulk ← true;
        re.unitofMeasure ← <sr.QohUnit>;
        INSERT the bulk resource entries re to the ResourceType table;
    else
        if sr has Type Properties then
            re.isIdentifiable←true;
            INSERT the identifiable resource entries re to the ResourceType table;
        else
            INSERT the unique identifiable resource entries re to the ResourceType table;
        end
    end
    foreach Property p of Resource sr do
        ae.Name ← <p.Name>;
        ae.Datatype ← <p.Type>;
        INSERT attribute entries ae to the Attribute table;
        rhaa.ResourceType_Id ← re.Id;
        rhaa.Attribute_Id ← ae.Id;
        rhaa.IsOptional ← false;
        rhaa.IsBulkProperty ← false;
        if p is Type Property then
            rhaa.IsBulkProperty ← true;
        end
        INSERT rhaa to the ResourceType_Has_AdditionalAttribute table;
    end
end
```

Algorithm 6.1: Resource Mapping Rules Algorithm

6.3 Mapping Agents to the REA DB

To illustrate the mapping of REA-DSL agents, the car producer example will be used again. As indicated by Figure 4.2 on page 41, agents can be distinguished into inside- (white-headed stick figures) and outside agents (black-headed stick figures). Similar to resources, they can have object- (green compartments) as well as type properties (yellow compartments). The affected tables by the agent mapping are **AgentType**, **Attribute**, and **AgentType_Has_AdditionalAttribute**.

The mapping of agents to the REA DB is illustrated in Figure 6.2. These rules are also shown in Algorithm 6.2 on page 63. Basically agents are represented as entries in the **AgentType** table

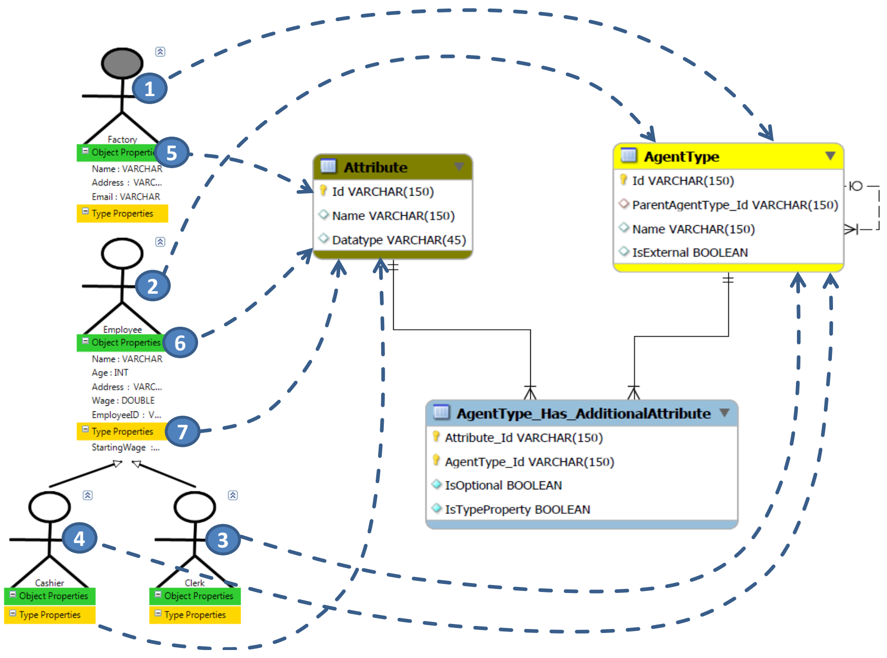


Figure 6.2: Agent Mapping

with their `Name`. `IsExternal` is used to differentiate inside- from outside agents. Therefore in the table entry for the external agent factory (1) `IsExternal` is true. For internal agents like employee (2) this value is set to false. Agents can have supertypes which is represented by the `ParentAgentType_Id` value. Examples in Figure 6.2 are clerks (3) and cashiers (4). Mapping agent properties to the REA DB works the same way as for resources. They are mapped to the **Attribute** table with their specified `Name` and `Datatype`. **AgentType_Has_-AdditionalAttribute** binds the properties to their agenttypes. In case of type properties like the startingwage (7), `IsTypeProperty` is set to true. For object properties like the factories address (5) or the employees actual wage (6) the value is set to false. Properties of agents that are inherited from supertypes have to be considered as well. The object properties of employees (5) are inherited by all subtypes (clerk (3) and cashier (4)). Every property of agents is saved once as entry in the **Attribute** table, but can be related to more than one agent in the **Agenttype_Has_-AddtionalAttribute** table. E.g., the clerk (3) agenttype is related to object- and type properties of its supertype (6,7). Since all properties that were specified in the REA-DSL agent model are not optional at the time when this thesis was written, `IsOptional` is false for all entries in the table.

6.4 Mapping the Operational/Planning View to the REA DB

As mentioned before, the operational view is derived from the planning view. Therefore, we describe no explicit mapping for the operational view. The value activities that are defined in the REA-DSL value chain view are also not explicitly mapped to the REA DB since they

```
input  : REA-DSL agent model $rdsl\_am$
output : Updated REA DB after inserting agenttype entries $ae$, attribute entries $attre$, and
         agenttype_has_additionalattribute entries $ahaa$
1  foreach Agent $sa$ in $rdsl\_am$ do
       // setting attribute values for agenttype entries
2      $ae$.Name ← <$sa$.Name>;
3      $ae$.isExternal ← false;
4      $ae$.superAgent ← <$sa$.Super>;
5      if $sa$ is an OutsideAgent then
6          $ae$.isExternal ← true;
7          INSERT the agent entries $ae$ to the AgentType table;
8      else
9          INSERT the agent entries $ae$ to the AgentType table;
10     end
11     foreach Property $p$ of Agent $sa$ do
12         $attre$.Name ← <$p$.Name>;
13         $attre$.Datatype ← <$p$.Type>;
14         INSERT attribute entries $attre$ to the Attribute table;
15         $ahaa$.AgentType_Id ← $ae$.Id;
16         $ahaa$.Attribute_Id ← $attre$.Id;
17         $ahaa$.IsOptional ← false;
18         $ahaa$.IsTypeProperty ← false;
19         if $p$ is Type Property then
20             $ahaa$.IsTypeProperty ← true;
21         end
22         INSERT $ahaa$ to the AgentType_Has_AdditionalAttribute table;
23     end
24 end
```

Algorithm 6.2: Agent Mapping Rules Algorithm

can be derived from the dualities. Instead all concepts of the defined planning view REA-DSL models, that are referenced in the value chain, are mapped to the database. These concepts include events/eventtypes, commitments, partcipation- and stockflow relationships, and defined properties. To demonstrate the mapping, the sales activity of the car producer is used (cf. Figure 4.4 on page 44). Since this task is quite extensive, the mapping will at first be explained for events/eventtypes, commitments, and the participation/stockflow relationships. Afterwards the mapping for the various properties will be described. The rules for mapping REA-DSL planning view models are illustrated in Algorithm 6.3 to 6.6. A description of the basic planning mapping (dualitytypes, contracttypes, and eventtypes) is described in Algorithm 6.4. These rules are extended by the additional attributes for the eventtypes in Algorithm 6.5. The rules for stockflows, participation relationships, and the respective properties is stated in Algorithm 6.6 and 6.3.

Mapping of Events/Eventtypes, Commitments, and Relationships

The tables of the REA DB, that are needed for the mapping are **ContractType**, **DualityType**, **ContractType_Allows_DualityType**, **EventType**, **DualityType_Allows_EventType**, **Event-TypeStockflow**, and **EventTypeParticipation**. Agents and resources are just referenced in the planning- or operational views of the REA-DSL. Therefore existing database entries for agents and resources (based on the aforementioned mappings in Chapter 6.2 and 6.3) are also reused.

Figure 6.3 illustrates how the mapping of a REA-DSL planning view model works in detail.

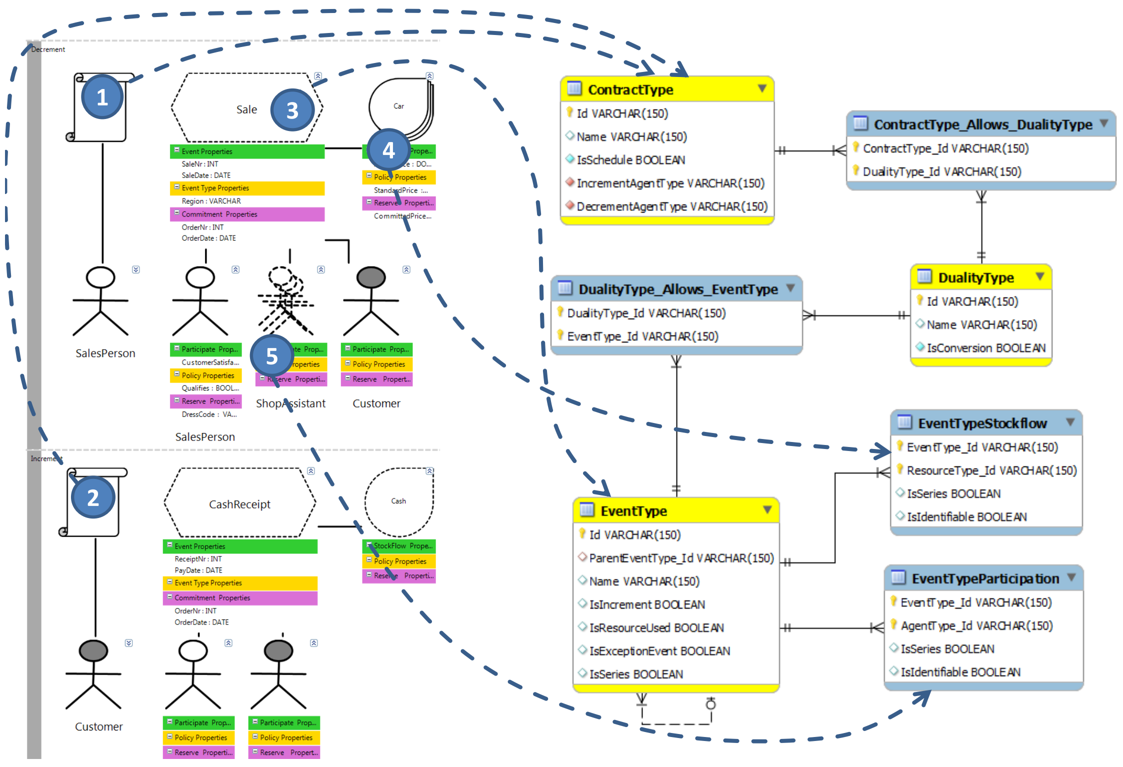


Figure 6.3: Event-, Commitment, and Relationship Mapping

The left side shows the sales activity, while the right side depicts the REA DB tables that are used to store the concepts. The sales activity represents a duality between a decrement (sales eventtype) and an increment event set (cashreceipt eventtype). Therefore an entry in the **DualityType** table is created with a `Name` and an `IsConversion` value set to false since the sales activity is an exchange. The decrement (1) and increment commitments (2), that are made to fulfill the events, are mapped to the **ContractType** table as one single entry, defining a contract with its `Name`. The `IsSchedule` value is set to false because exchanges are planned in form of a contract and not a schedule. Agents that are legally committing to fulfill the decrement and increment event sets are stored as `IncrementAgentType` (customer) and `DecrementAgentType` (salesperson). The relationship between dualitytypes and their corresponding contracttypes is represented in the **ContractType_Allows_DualityType** table by saving their `Ids`. Events or eventtypes are stored in the **EventType** table with the `Name` and, if an event hierarchy exists, the `ParentEventType_Id` (this functionality is currently not supported by REA-DSL though). For the sale event (3) no parent element exists, but the `IsIncrement` and `IsResourceUsed` value is set to false, because during a sale the resource is decremented and not used. `IsSeries` is also false since a sale is not a resource series. Stockflows are mapped to the **EventtypeStockflow** table. If the affected resource is a series, `IsSeries` is true. `IsIdentifiable` is true when resourcetypes are affected. Since during the sale event a car resourceseries is decremented (4), this stockflow is represented with `IsSeries` as true and `IsIdentifiable` as false. The mapping of participations follows the same principle. The agenttypeseries that participates in the sale event (5) is depicted in the **EventtypeParticipation** table with `IsSeries` and `IsIdentifiable` values both true.

Mapping of Properties

Properties that were specified in the planning view are saved to the **Attribute** table of the REA DB. Compared to the resource- and agent views, not only two kinds of properties exist. Event-, event type-, and commitment properties can be specified for events/eventtypes. Stockflow relationships can have stockflow-, policy-, and reserve properties. Participate relationships can contain participate-, policy-, and reserve properties. The **EventTypeStockflow_Has_AdditionalAttribute**, **EventType_Has_AdditionalAttribute**, and **EventTypeParticipation_Has_AdditionalAttribute** tables are affected by the property mapping. These tables distinguish between the aforementioned property types and relate attributes to events or eventtypes, stockflows, and participations.

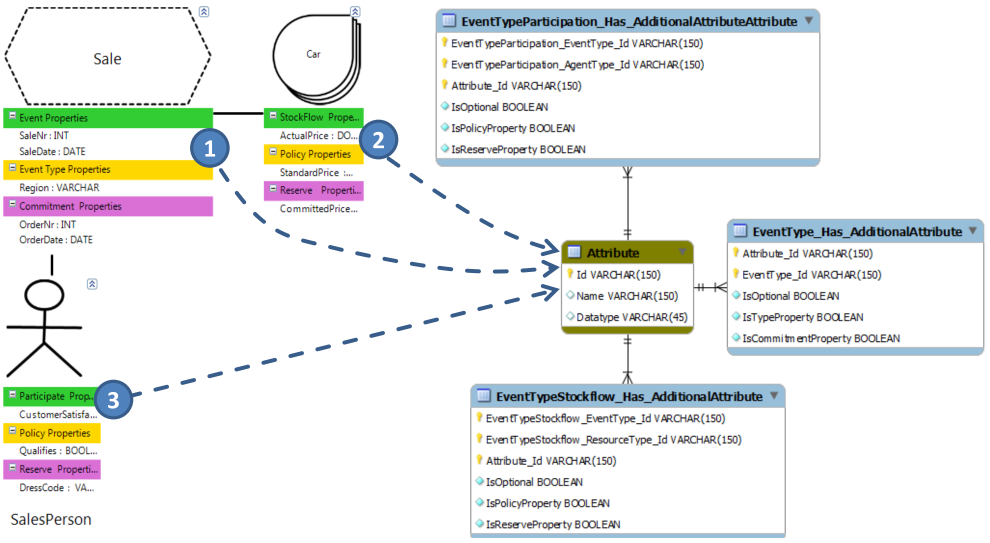


Figure 6.4: Planning View Attribute Mapping

Figure 6.4 shows an excerpt of the car producers sales activity to describe the mapping of all relevant properties. Properties of the sale eventtype (1) are stored as **Attribute** entries. The `Names` of these properties and their `Datatype` correspond to the defined values in the REA-DSL. Therefore saleNr, saleDate, region, orderNr and orderData are all inserted as attributes. The distinction between the different property types of events/eventtypes is made in the **EventType_-Has_Additional_Attribute** table. For example the region property's `Id` in the **Attribute** table is stored together with the `Id` of the sale eventtype in **EventType_Has_Additional_Attribute**. Since region is an event type property, the `IsTypeProperty` value is set to true. For the orderNr and orderDate of the sale eventtype (1) only `IsCommitmentProperty` is set to true, to identify them as commitment properties. The mapping of the participate- and stockflow properties works similar. ActualPrice, standardPrice, and committedPrice of the stockflow between the car resourceseries and the sale eventtype (2) are stored as entries in the **Attribute** table. Customersatisfaction, qualifies, and dresscode of the participate relationship (3) are saved in that table as well. **EventtypeStockflow_Has_AdditionalAttribute** and **EventtypeParticipation_-Has_AdditionalAttribute** relate the aforementioned properties to the corresponding eventtype. `IsPolicyProperty` and `IsReserveProperty` are used to differentiate between policy- and reserve properties, by defining one of the values as true. If both are false, the property either is a stockflow or a participate property. E.g., actualPrice of the stockflow relation (2) is mapped to the **Attribute** table with its defined `Name` and `Datatype`. Since actualPrice is a stockflow property, `IsReserveProperty` and `IsPolicyProperty` are both false in the **EventtypeStockflow_Has_AdditionalAttribute** table.

```
input  : Eventtype $et$
output : Updated REA DB after inserting eventtypeparticipationentries $pte$, attribute entries $attre$, and
         eventtypeparticipation_has_additionalattribute entries $etphaa$
1  foreach Agent a of Eventtype Resources et.Agents do
       // inserting participations
2      $pte$.IsAgentIdentifiable ← false;
3      $pte$.IsAgentSeries ← false;
4      if $a$ is AgentType or $a$ is AgentTypeSeries then
5          $pte$.IsIdentifiable ← true;
6      end
7      if $a$ is AgentSeries or $r$ is AgentTypeSeries then
8          $pte$.IsSeries ← true;
9      end
10     INSERT $pte$ to the EventTypeParticipation table;
       // inserting participation propertes
11     foreach Property p of EventTypeParticipation do
12         $attre$.Name ← <$p$.Name>;
13         $attre$.Datatype ← <$p$.Type>;
14         INSERT attribute entries $attre$ to the Attribute table;
15         $etphaa$.EventTypeParticipation_Id ← $pte$.Id;
16         $etphaa$.Attribute_Id ← $attre$.Id;
17         $etphaa$.IsOptional ← false;
18         $etphaa$.IsPolicyProperty ← false;
19         $etphaa$.IsReservementProperty ← false;
20         if p is Policy Property then
21             $etphaa$.IsPolicyProperty ← true;
22         end
23         if p is Reserve Property then
24             $etphaa$.IsReserveProperty ← true;
25         end
26         INSERT $etphaa$ to the EventTypeParticipation_Has_AdditionalAttribute table;
27     end
28 end
```

Algorithm 6.3: Mapping Rules Algorithm for Participations and Attributes

```
input  : REA-DSL value chain model $rdsl\_vcm$
output : Updated REA DB after inserting contracttype entries $cte$, contracttype_allows_dualitytype entries $ctadt$,
         dualitytype entries $dte$, eventtype entries $ete$, dualitytype_allows_eventtype entries $dtaet$
foreach Business activity ba in $rdsl\_vcm$ do
    $dte$.Name ← <$ba$.Name>;
    $dte$.IsConversion ← false;
    if ba.Type is Conversion then
        $dte$.IsConversion ← true;
    end
    INSERT the dualitytype entries $dte$ to the DualityType table;
    if ba defines Contract then
        $cte$.IsSchedule ← false;
        $cte$.IncrementAgentType ← $ba$.IncrementSet.Commitment.Agent;
        $cte$.DecrementAgentType ← $ba$.DecrementSet.Commitment.Agent;
        INSERT the contracttype entries $cte$ to the ContractType table;
        $ctadt$.ContractType_Id ← $cte$.Id;
        $ctadt$.DualityType_Id ← $dte$.Id;
        INSERT $ctadt$ to the Contracttype_Allows_Dualitytype table;
    end
    foreach Eventtype e of Business activity ba do
        $ete$.Name ← <$e$.Name>;
        $ete$.IsIncrement ← false;
        $ete$.IsUsed ← false;
        $ete$.IsSeries ← false;
        if e is Íncrement Event then
            $ete$.IsIncrement ← true;
        end
        foreach Resource r in e.Resources do
            if r is Used then
                $ete$.IsUsed ← true;
            end
        end
        if e is EventSeries or e is EventTypeSeries then
            $ete$.IsSeries ← true;
        end
        INSERT $ete$ to the EventType table;
        $dtaet$.DualityType_Id ← $dte$.Id;
        $dtaet$.EventType_Id ← $ete$.Id;
        INSERT $dtaet$ to the DualityType_Allows_Eventtype table;
    end
end
```

Algorithm 6.4: Planning Mapping Rules Algorithm

```
input  : Eventtype $et$
output : Updated REA DB after inserting attribute entries $attre$, and eventtype_has_additionalattribute entries $ethaa$
1  foreach Property p of Eventtype et do
       // setting attribute values for additional attribute entries of event et
2      $attre$.Name ← <$p$.Name>;
3      $attre$.Datatype ← <$p$.Type>;
4      INSERT attribute entries $attre$ to the Attribute table;
5      $ethaa$.EventType_Id ← $ete$.Id;
6      $ethaa$.Attribute_Id ← $attre$.Id;
7      $ethaa$.IsOptional ← false;
8      $ethaa$.IsTypeProperty ← false;
9      $ethaa$.IsCommitmentProperty ← false;
10     if p is Type Property then
11         $ethaa$.IsTypeProperty ← true;
12     end
13     if p isCommitment Property then
14         $ethaa$.IsCommitmentProperty ← true;
15     end
16     INSERT $ethaa$ to the EventType_Has_AdditionalAttribute table;
17 end
```

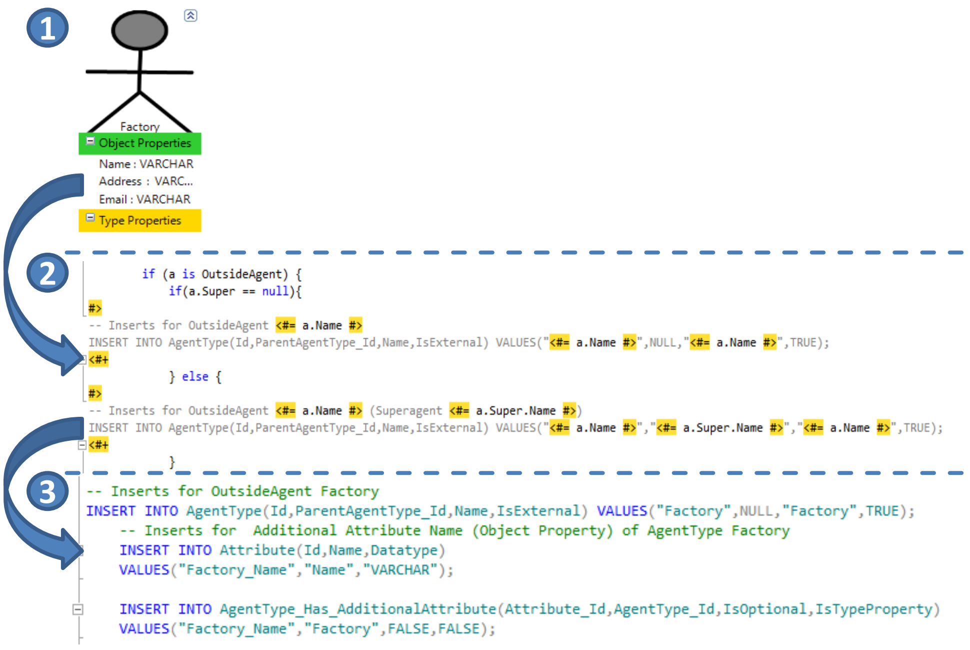
Algorithm 6.5: Mapping Rules Algorithm for Additional Attributes of Events

```
input  : Eventtype $et$
output : Updated REA DB after inserting eventtypestockflow entries $sfte$, attribute entries $attre$,and
         eventtypestockflow_has_additionalattribute entries $etsfhaa$
1  foreach Resource r of Eventtype Resources et.Resources do
       // inserting stockflows
2      $sfte$.IsIdentifiable ← false;
3      $sfte$.IsSeries ← false;
4      if r is ResourceType or r is ResourceTypeSeries then
5          $sfte$.IsIdentifiable ← true;
6      end
7      if r is ResourceSeries or r is ResourceTypeSeries then
8          $sfte$.IsSeries ← true;
9      end
10     INSERT $sfte$ to the EventTypeStockflow table;
       // inserting stockflow propertes
11     foreach Property p of EventTypeStockflow do
12         $attre$.Name ← <$p$.Name>;
13         $attre$.Datatype ← <$p$.Type>;
14         INSERT attribute entries $attre$ to the Attribute table;
15         $etsfhaa$.EventTypeStockflow_Id ← $sfte$.Id;
16         $etsfhaa$.Attribute_Id ← $attre$.Id;
17         $etsfhaa$.IsOptional ← false;
18         $etsfhaa$.IsPolicyProperty ← false;
19         $etsfhaa$.IsReserveProperty ← false;
20         if p is Policy Property then
21             $etsfhaa$.IsPolicyProperty ← true;
22         end
23         if p is Reserve Property then
24             $etsfhaa$.IsReserveProperty ← true;
25         end
26         INSERT $etsfhaa$ to the EventTypeStockflow_Has_AdditionalAttribute table;
27     end
28 end
```

Algorithm 6.6: Mapping Rules Algorithm for Stockflows and Attributes

6.5 Generating Insert Statements for the REA DB

Based on the predefined mapping, SQL inserts are generated for the REA-DSL models to save them to the REA DB. Following the same approach as Mayrhofer [41] while developing the REA-DSL, Microsoft's T4 text template transformation toolkit (T4) [7] is used to accomplish this task. The T4 toolkit is a model-to-code transformation tool where text templates are used to generate text files. These templates contain directives specifying how it is processed, text blocks that are copied to the output without further changes, and control blocks in form of C# or Visual Studio .NET program code for conditional or repeated text parts. Furthermore, the code blocks can reference elements of DSL models. Therefore the elements of the REA-DSL models can be used and further processed in the text templates. The generated file can be text of any kind. It is possible to generate HTML files as well as SQL files, as needed for the purpose of this thesis.



```
        if (a is OutsideAgent) {
            if(a.Super == null){
#>
-- Inserts for OutsideAgent <#= a.Name #>
INSERT INTO AgentType(Id,ParentAgentType_Id,Name,IsExternal) VALUES("<#= a.Name #>",NULL,"<#= a.Name #>",TRUE);
<#+
            } else {
#>
-- Inserts for OutsideAgent <#= a.Name #> (Superagent <#= a.Super.Name #>)
INSERT INTO AgentType(Id,ParentAgentType_Id,Name,IsExternal) VALUES("<#= a.Name #>","<#= a.Super.Name #>","<#= a.Name #>",TRUE);
<#+
            }
```

```
-- Inserts for OutsideAgent Factory
INSERT INTO AgentType(Id,ParentAgentType_Id,Name,IsExternal) VALUES("Factory",NULL,"Factory",TRUE);
    -- Inserts for  Additional Attribute Name (Object Property) of AgentType Factory
    INSERT INTO Attribute(Id,Name,Datatype)
    VALUES("Factory_Name","Name","VARCHAR");

    INSERT INTO AgentType_Has_AdditionalAttribute(Attribute_Id,AgentType_Id,IsOptional,IsTypeProperty)
    VALUES("Factory_Name","Factory",FALSE,FALSE);
```

Figure 6.5: T4 Text Templates and generated SQL Inserts for an Agent

Figure 6.5 illustrates how SQL insert statements are generated for an agent in the REA-DSL model. The outside agent `factory` has three additionally defined object properties: `name`, `address`, and `email` (1). T4 text templates define how agents are transformed to insert statements. Elements of the REA-DSL model can be referenced by variable names. In Figure 6.5, "`a`" is an agent element in the REA-DSL model. Depending on the fact if this element is an inside- or outside agent and whether superagents exists, insert statements with differing values are generated (2). Since `factory` is an outside agent that has no parent agents (cf. Figure 4.2 on page 41), the value of `ParentAgentType_Id` is null and `IsExternal` is true. Generated inserts for the object properties of the factory are also shown. One statement adds an attribute while the other statement relates the property to the agenttype (3). The T4 transformation from additional properties to SQL inserts is not shown in Figure 6.5. The created statements are included for demonstration purpose to indicate the capabilities of the T4 text template transformation toolkit. The generated SQL insert statements need to be loaded into a relational data modeling tool. Using MySQL Workbench with an already running REA DB enables a population of the database by simply executing the generated scripts.

CHAPTER 7

Generating User Interfaces based on Business Models

7.1 Overview

Chapter 6 illustrated how to generate SQL insert scripts that can be used to save REA-DSL business models in the REA DB. However, these models just represent a blueprint for business events that can occur. In order to conduct actual business cases, a user interface has to be generated based on the business model data that allows the specification of values for actual events and saving them to the REA DB. Figure 7.1 shows this concept. The saved REA-DSL models, consisting of agent-, resource-, value chain-, operational-, and planning views, are represented as business model data in the REA DB and used for the generation of adapted user interfaces (1). This masks can then be used to define actual business cases that are stored in the generic part of the database (2).

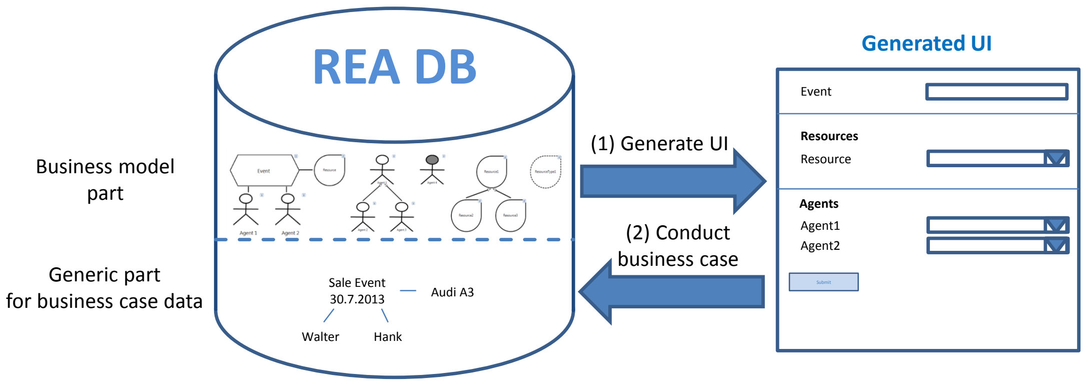


Figure 7.1: Concept of the UI Generation

7.2 Relevant REA DB Parts for administrating Business Cases

Before going any further, the relevant tables of the REA DB's generic part that are needed to save business cases have to be identified. As briefly stated in Chapter 5, the REA-Core subpart contains the majority of the needed tables. A subset of AdditionalAttributes is necessary to store actual values of defined attributes.

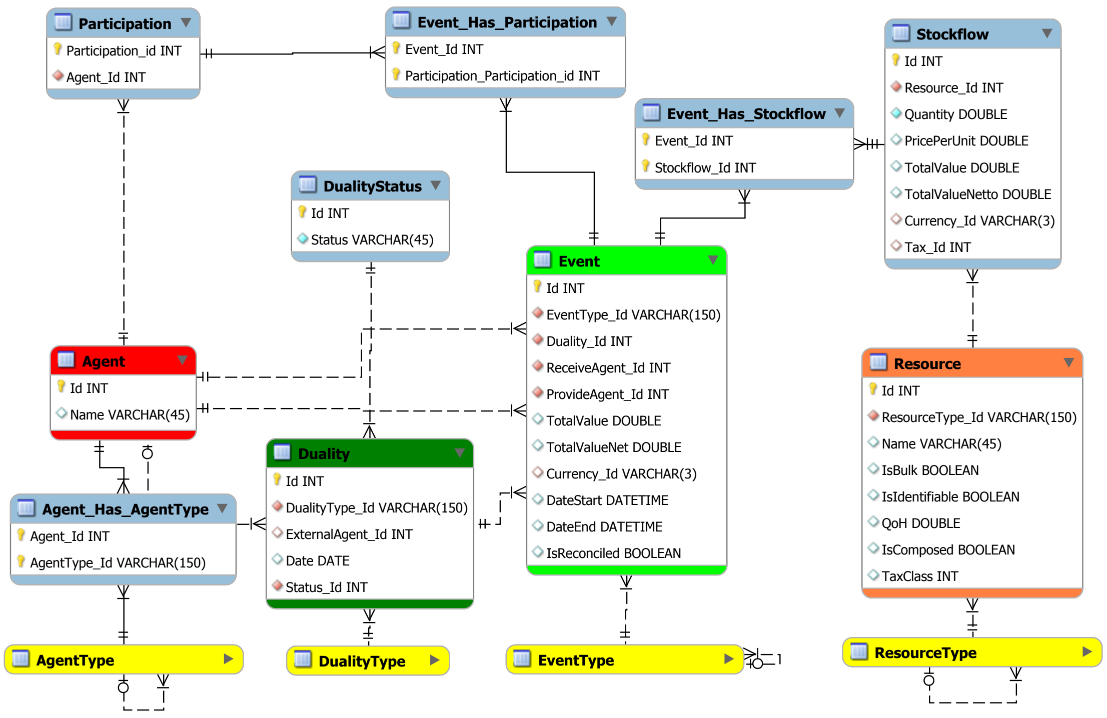


Figure 7.2: REA Core Subset relevant for Business Cases

Figure 7.2 shows the tables of REA-Core with the tables that are needed to represent a business case (with visible attributes). Concrete agents, resources, and dualitystatuses are saved in the **Agent**, **Resource**, and **DualityStatus** tables. While agents and dualitystatuses only possess `Ids` and `Names`, resources also have specified values for `IsBulk`, `IsIdentifiable`, and `QoH`. When an actual business case is specified, it is saved as an entry in the **Duality** table, referencing a predefined status that already exists in the REA DB. The `Date` value represents the creation date. Since a duality consists of several events, each one is represented as an entry of the **Event** table with its referenced `Duality_Id`, `Eventtype_Id`, `DateStart`, and `DateEnd`. The values for `ProvideAgent` and `ReceiveAgent` need to reference agents that are already saved. Stockflows and participations that belong to an event are saved as entries in the **Stockflow** and **Participation** tables with specified values and referenced agents or resources. A concrete business case is always based on an underlying business model which defines its structure. In Figure 7.2 this fact is illustrated by the **Dualitytype**, **Agenttype**, **Eventtype**, and **Resourcetype** tables that hold the business model data (cf. Chapter 6). The creation of a business case is therefore restricted to the activities that occur in the business model (only dualitytypes that are defined by the model can be referenced when inserting values to the duality table). Furthermore, the structure of events is predefined since dualitytypes only allow certain eventtypes. The same holds for participating agents and affected resources that only allow agents/resources of a certain type. Another important thing that has to be mentioned in this context are properties (e.g., object- or event properties) that are defined on business model elements (e.g., agents, resources, or events). A REA-DSL business model only defines that such properties exist and which datatype they possess. For a business case the actual values of these properties also need to be persisted in the database.

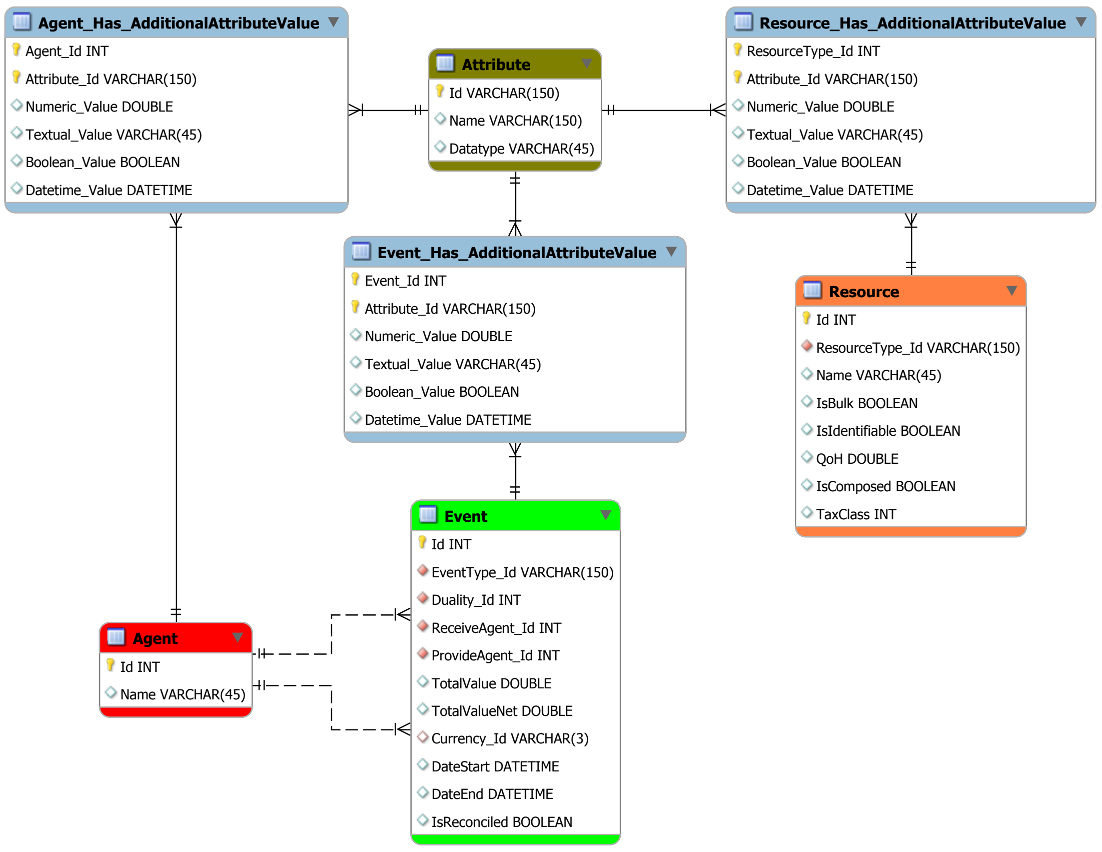


Figure 7.3: Values of Additional Attributes for Business Cases

Figure 7.3 shows the tables that are necessary to store property values. Additional attributes for agents are saved in the **Agent_Has_AdditionalAttributeValue** table. The values can either be `Numeric`, `Textual`, `Boolean-` or `Datetime_Values`. For events and resources the same approach is used by saving the data in the **Event_Has_AdditionalAttributeValue** or **Resource_Has_AdditionalAttributeValue** table. The attributes that have to be defined for entities are defined in the business model. For example when a new agent is saved, its agenttype (the entry in the REA DB business model part) has to be referenced. If additional properties are defined for that agenttype, the actual agent has to define values for them. These values are also restricted by the business model. If a REA-DSL model specifies an agent with an object property salary that has a double datatype, it is only possible to define the attribute value for the corresponding property as numeric.

7.3 Scope of the UI-Generation Prototype

In order to support users in conducting business cases and persist them in the REA DB, we provide a prototype that generates user interfaces adapted to the already stored business model data. As already mentioned, specifications of the underlying model have to be represented in the generated masks. Since already existing dualitystatus, agent, and resource records are a mandatory requirement to specify business cases, the prototype has to provide the possibility to create, update, or delete those entities. Users should only be able to create dualities for activities that occur in the value chain of a stored business model. The status of a duality can be chosen from an already persisted set of statuses in the REA DB. Depending on the selected dualitytype, the number of events that need to be specified is predefined. For each event provide- and receive agents have to be declared. The selection possibilities for these agents are already restricted by the business model since only specific types are acceptable. The number of participating agents per event is also represented in the model. Therefore, the generated user interface has to assure that all needed participation relationships can be specified by the user. Since participating agenttypes in an event are stated in the business model, the actual agents that can be chosen by the user are restricted. The same conditions have to hold for stockflow relationships and affected resources. A special case are participating agent(type)series and affected resource(type)series. When such elements are defined, it depicts that at least one element is related to an event. For example a participating agentseries illustrates that at least one agent of the corresponding type participates in the event. But there also can be more participants (of the same type). The prototype therefore has to grant users the opportunity to also specify such cases.

The intended functionality of the UI-Generation prototype is summarized in the following and can be divided in two sections: the administration of all entities that are needed to specify business events (1-3) and the actual business case creation (4-8) .

1. **Administrating dualitystatuses.** Create, update, and delete records in the dualitystatus table of the REA DB. Before saving, user inputs should be validated to prevent errors.

2. **Administrating agents.** Create, update, and delete agents. Since agents need to reference an existing agenttype, all existing types in the REA DB can be chosen by the user. Depending on that choice, the additional attributes are listed and have to be specified. All user inputs need to be validated.

3. **Administrating resources.** Create, update, and delete resources. Resourcetypes are chosen from existing records that were persisted as business model data. Similar to agents, additional attribute values need to be defined and inputs are validated.

4. **Creating business cases based for specific dualitytypes.** Business cases can be specified for all value activities that occur in the underlying business models value chain. Since these activities are represented as entries of the dualitytype table, the prototype must allow users to choose one of the existing entries. Based on that selection, masks for the detailed specification of events are shown.

5. **Definition of events.** The number of events that occur in a business case is defined in the planning- or dualityviews of a REA business model. When saved to the REA DB, these events are persisted in the eventtype table. The prototype is capable of automatically visualizing masks for all events that belong to the selected dualitytype. These masks should encompass elements to define everything that is needed to save an event in the REA DB (e.g., provide- and reveive agent, start- and enddate). Furthermore, it has to be possible to specify values for additional attributes that are derivated from the business models. All inputs should be accordingly validated.

6. **Definition of participations.** The number of participating agents is already reflected in the saved business model data. The prototype should list all participations in the user interface for each event and enable users to specify actual agents. Only agents that have the same type as specified in the business model can be selected. If participation relationships possess additional attributes, input fields should be generated to define the values. Participating agent(type)series also need to be considered. It therefore has to be possible to define further participations if a series occurs.

7. **Definition of stockflows.** The interface for an event should encompass a section to define all stockflows that were specified in the business model. The concrete resources have to be selected by the users. Again only resources that have the corresponding types are possible. Similar to participations, the definition of affected resource(type)series during stockflows has to be considered.

8. **Saving business cases to the REA DB.** After users filled in all the information that is necessary, the business case should be persisted in the database without errors.

   As illustrated, the generation of sections that can be used to specify commitments is not included in the prototypes functionality. While the main goal of this thesis is to specify a mapping to insert REA-DSL models to the REA DB, the UI-Generation prototype is created to prove that the saved business model data can be used as foundation for adaptable user interface creation. The masks for commitments basically would look similar to those of actual events, but including them would not have provided us with additional insights on how to generate user interfaces based on the persisted business models. Therefore this part was excluded from the prototype's scope.

7.4 Architecture and used Technologies

Figure 7.4 shows the technologies that are used to develop the UI-Generation prototype.

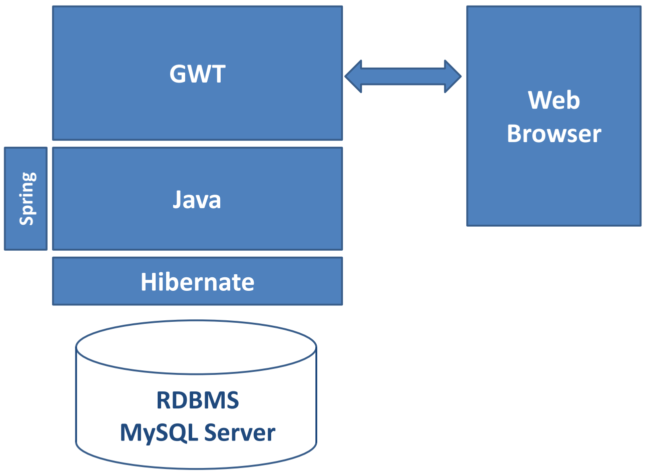


Figure 7.4: Used Technologies

MySQL

The foundation of the UI-Generation prototype is the database with the persisted business models. As already mentioned in Chapter 6, MySQL Workbench was used to design the database. We therefore stick to that decision and use MySQL as the database management system. A great advantage in doing so is that the database schema of the REA DB can directly be adapted using MySQL Workbench. The designed database can then be easily forward engineered and hosted with MySQL Server. Furthermore, the SQL insert scripts that were generated for REA-DSL business models can easily be loaded and executed to quickly populate the database.

Google Web Toolkit

For the creation of the user interfaces based on saved business models in the database, the Google Web Toolkit (GWT) will be used. GWT is a toolkit for the development of web applications, that possesses a Java to JavaScript compiler, a XML-Parser, internationalization support, and an integration of the JUnit testing framework[1]. GWT also specifies an interface for asynchronous calls to web servers using remote procedure calls (RPC) or JavaScript object notation (JSON) . Since the long term goal of the REAlist project is an ERP system that can be hosted in the cloud and used by tenants with a web browser, creating the prototype with a tool capable of this made sense. The outstanding advantage of GWT is that due to the Java to JavaScript cross compiler client- and server side code can be entirely written in Java. Therefore, a variety of well known frameworks can be used.

[1]`http://junit.org/`

Hibernate and Hibernate Tools

Hibernate[2] is a persistence framework for Java. It supports object relational mapping (ORM), allowing to save Java objects in a relational database. Furthermore, retrieved records from the database are transformed to objects and can be used in the Java application. Hibernate is compatible with several databases. Therefore, we use it to extract and store data in the MySQL REA DB. As illustrated in Chapter 5 and Appendix A, the REA DB consists of a lot of tables. Manually coding Java classes for all subparts of the database that are needed for the prototype creation would be cumbersome and error prone. Therefore we use Hibernate Tools[3] to reverse engineer the existing database and automatically create domain model classes, hibernate mapping files, and annotated EJB3 entity beans. During development, the data structure of the REA DB changed several times. Therefore Hibernate Tools proved very useful to quickly generate domain classes after adaptations.

Spring

Another useful framework for Java platforms is Spring[4]. It provides a programming and configuration model for Java enterprise applications. One of the core characteristics of the Spring framework is the support of dependency injection.

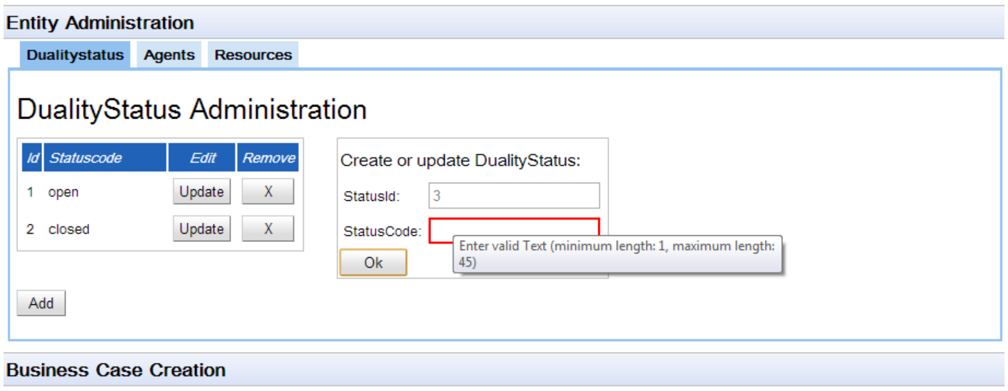
7.5 Administration of Entities

The functionality of the prototype will be explained by the business model of the car producer that was already used to explain the REA-DSL and the mappings to the REA DB. It consists of a resource view defining all resources, an agent view specifying all agents, a value chain view, operational-, and planning views (cf. Chapter 4). When creating a business case for this model, resources, agents, and dualitystatuses already have to be persisted in the REA DB since events, participations, and stockflows just reference these elements. Therefore, the prototype provides an administration section that enables users to create these entities.

Dualitystatuses

Figure 7.5 shows the generated interface for the administration of dualitystatuses. Statuses that are already saved in the REA DB are listed in the table on the left and can be updated or deleted. When a new status is added to the database, `id's` are generated automatically. However, the `statuscode` has to be specified by the user. Since this value is mandatory and is stored as VARCHAR with 45 characters at most (cf. Figure 7.2 on page 72), textboxes can be validated for correctness before something is saved or updated. Validation errors display notifications to the user. The generated user interface for the administration of dualitystatuses is not complex. Statuses can not possess additional attributes and only fields for fixed attributes (in this case the statuscode of the duality) are generated.

[2]`http://hibernate.org/`
[3]`http://hibernate.org/tools/`
[4]`http://projects.spring.io/spring-framework/`


Entity Administration

Dualitystatus | Agents | Resources

DualityStatus Administration

Id	Statuscode	Edit	Remove
1	open	Update	X
2	closed	Update	X

Add

Create or update DualityStatus:

StatusId:	3
StatusCode:	

Enter valid Text (minimum length: 1, maximum length: 45)

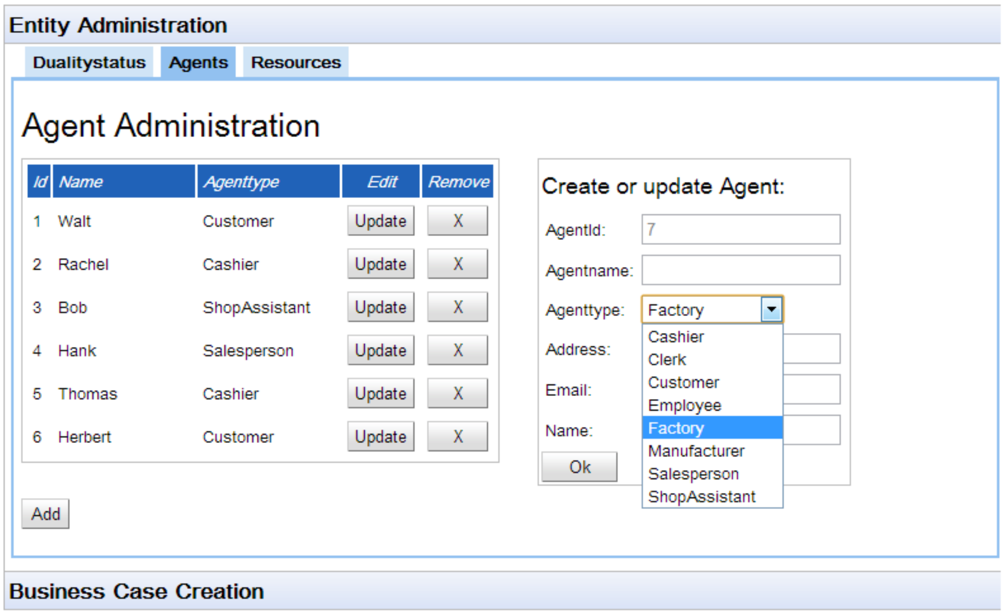
Ok

Business Case Creation

Figure 7.5: Administrating Dualitystatuses

Agents

Figure 7.6 illustrates how the interface for the agent administration looks like.


Entity Administration

Dualitystatus | Agents | Resources

Agent Administration

Id	Name	Agenttype	Edit	Remove
1	Walt	Customer	Update	X
2	Rachel	Cashier	Update	X
3	Bob	ShopAssistant	Update	X
4	Hank	Salesperson	Update	X
5	Thomas	Cashier	Update	X
6	Herbert	Customer	Update	X

Add

Create or update Agent:

AgentId:	7
Agentname:	
Agenttype:	Factory
Address:	
Email:	
Name:	

Cashier
Clerk
Customer
Employee
Factory
Manufacturer
Salesperson
ShopAssistant

Ok

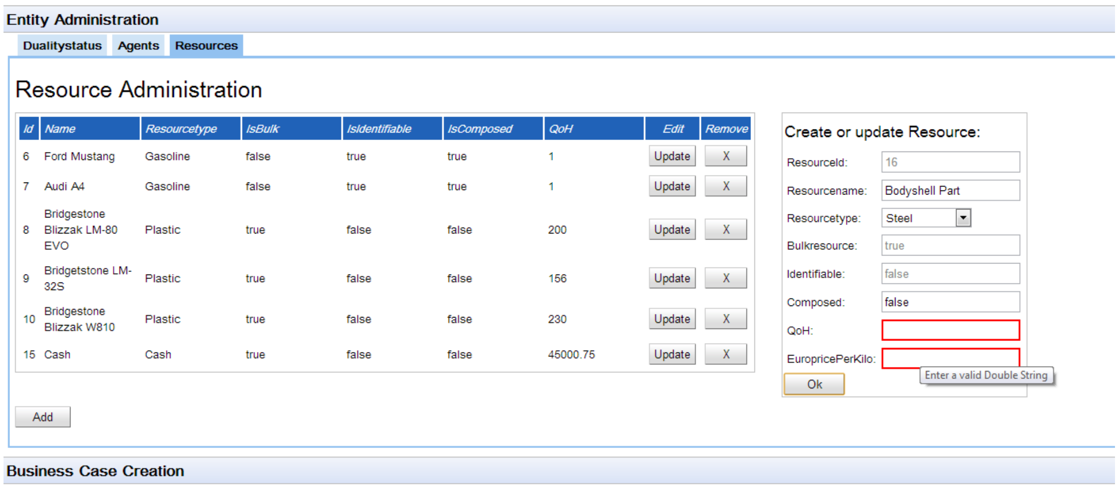
Business Case Creation

Figure 7.6: Administrating Agents

Basically the UI looks similar to the administration of dualitystatuses. A table lists all entities that are already persisted in the database and new agents can be created. Agents have an `id` that is automatically generated when a record is saved and a name that has to be specified. Besides that, each agent has an `agenttype` that is selected by the user. The possible manifestations are already persisted in the REA DB and are derived from the saved agent view model of the car producer (cf. Figure 4.2 on page 41). Furthermore, labels and textboxes for the definition of attribute values are generated. For the selected agenttype (factory) these attributes are `name`, `address`, and `email`. Each input field for attribute values is also validated before saving as will be shown for resources in the next section.

Resources

The administration of resources is stated in Figure 7.7. For demonstration purposes several resources were already created and are listed in the table to the left. The right side shows the creation of a new resource where values for the fixed attributes (`name` and `composed`) have to be declared. Furthermore, an existing `resourcetype` has to be chosen. Similar to the agent administration, possible types are retrieved from the already stored business model data. Since the model of the car producer is used, those resource types are labor, car, gasoline, diesel, hybrid, cash, raw material, steel, plastic, aluminium, rubber, and glass (cf. Figure 4.1 on page 40). The `QoH` only is needed if the selected type is a bulk resource. Otherwise the textbox will not be visible. When a type possesses properties, values have to be declared. The resource that should be added in Figure 7.7 is a steel part. Therefore, the only additional property is `europricePerKilo`. When trying to save the resource, the prototype recognizes that user inputs were incorrect (QoH and europricePerKilo are undefined).



Resource Administration

Id	Name	Resourcetype	IsBulk	IsIdentifiable	IsComposed	QoH	Edit	Remove
6	Ford Mustang	Gasoline	false	true	true	1	Update	X
7	Audi A4	Gasoline	false	true	true	1	Update	X
8	Bridgestone Blizzak LM-80 EVO	Plastic	true	false	false	200	Update	X
9	Bridgetstone LM-32S	Plastic	true	false	false	156	Update	X
10	Bridgestone Blizzak W810	Plastic	true	false	false	230	Update	X
15	Cash	Cash	true	false	false	45000.75	Update	X

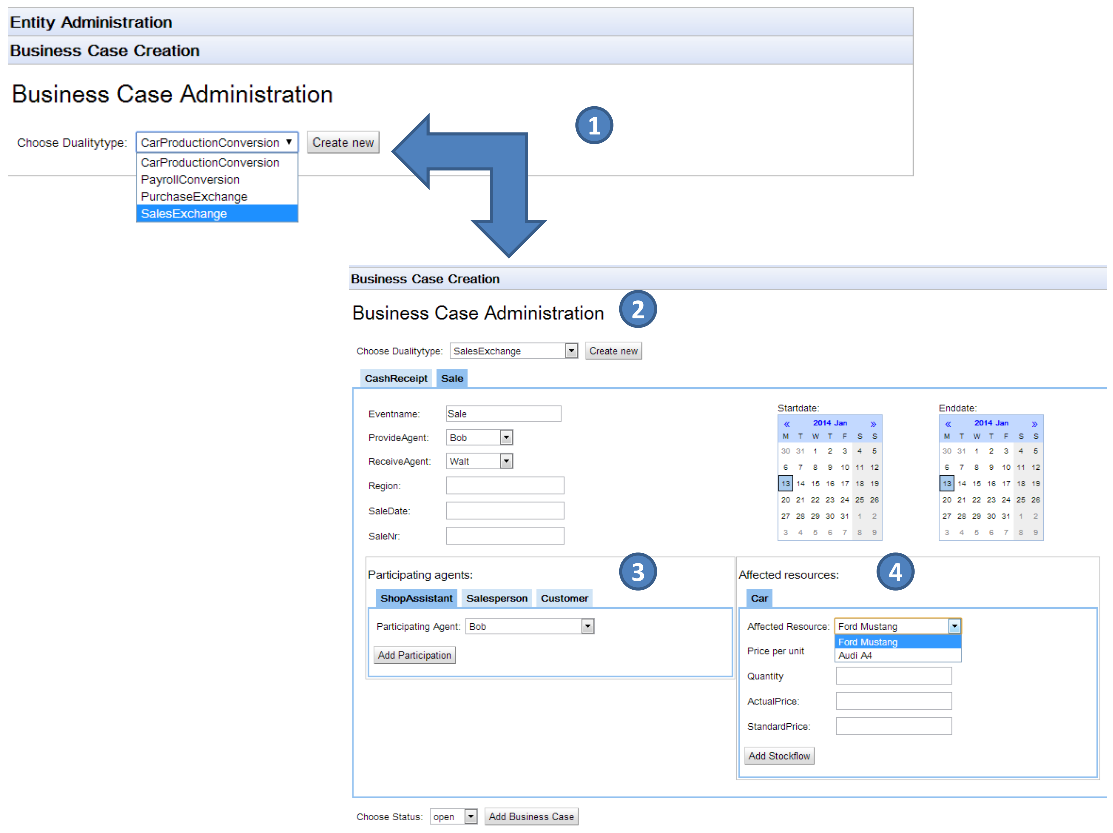
Create or update Resource:

ResourceId:	16
Resourcename:	Bodyshell Part
Resourcetype:	Steel
Bulkresource:	true
Identifiable:	false
Composed:	false
QoH:	
EuropricePerKilo:	

Figure 7.7: Administrating Resources

7.6 Conducting Business Cases

After inserting some records for dualitystatuses, agents, and resources, a concrete business case can be defined for an existing dualitytype. The types that can be chosen are already persisted in the business model data since each activity in the REA-DSL business models value chain is stored as dualitytype. For the fictitious car producer four such activities exist: purchase, carproduction, sales, and payroll (cf. Figure 4.3 on page 42). Therefore, the listbox in the section for business



Choose Dualitytype: CarProductionConversion

Choose Dualitytype: SalesExchange

Eventname: Sale
ProvideAgent: Bob
ReceiveAgent: Walt
Region:
SaleDate:
SaleNr:

Participating Agent: Bob

Affected Resource: Ford Mustang
Price per unit
Quantity
ActualPrice:
StandardPrice:

Choose Status: open

Figure 7.8: Sales Business Case Conduction

case creation of Figure 7.8 offers these choices to the user (1). Depending on the users selection, the masks for the detailed event definitions are shown. In Figure 7.8 the sales dualitytype (cf. Figure 4.4 on page 44) was selected leading to an event panel with two tabs that represent the events that have to be specified (sale and cashreceipt) (2). Since the sale event contains properties, textboxes are generated for each single one, that allow the specification of the additional attribute values. These attributes are `saleNr`, `saleDate`, and `region`. The remaining two (`orderNr` and `orderDate`) are omitted since commitment properties are related to the commitments and not the actual events. Since commitment definitions are not part of the UI-Generation prototype, no textboxes for these attributes are generated. Each event has `provide-` and `receive` agents that need to be defined. The prototype automatically recognizes which agents are qualified and proposes them to the user. For example when an activity is an exchange, only inside agents can provide resources during a decrement event (since a resource that has value to the company is decreased which can only be done by inside agents). On the other hand if the event is an increment, only external agents are providing while internal agents receive. The UI-Generation prototype only offers applicable choices to the user. Since an event has a `starting-` and `end date`, datepickers are generated that support users in the selection. Besides that, each event has agents that participate (3) and resources that are affected in the event (4). Participations and stockflows are also restricted by definitions in the business model, meaning that only specific persisted agents or resources can be chosen. During the sale event, the car resource is decreased. Therefore only resources that have resourcetype car (or one of its subtypes) can be selected. In Figure 7.8 Ford Mustang, Audi A8, and Audi A3 are possible choices. Since the stockflow decreasing a car is defined as series (cf. Figure 4.4 on page 44), users need to have the opportunity to define several cars that are affected during the event. In Figure 7.8, this is indicated by the additional button ("Add Stockflow") in the stockflow section (4). Participations for the sale event are generated the same way. Three participating agents exist: salesperson, shop assistant, and customer. For shop assistants additional participations can be created since it is defined as series in the business model. As explained in Chapter 7.3, the specification of commitments is not supported by the UI-Generation prototype.

CHAPTER 8

Evaluation of the Artifact

8.1 Basic Concept of the Artifact Evaluation

In order to evaluate the work that has been conducted in this thesis, it is necessary to test the functionality of the created artifact based on different scenarios. So far the business model of the fictitious car producer was used to explain the REA-DSL itself (cf. Chapter 4), the mapping to the REA DB (cf. Chapter 6), and the automated generation of user interfaces based on the saved data (cf. Chapter 7). The intention behind that was to use an ongoing example that comprises all possible concepts of the REA-DSL and facilitate readers of this thesis to follow the explanations. However, in order to prove that the created artifact works, it is necessary to evaluate its functionality and completeness for other business models as well.

When developing the REA-DSL, Dieter Mayrhofer used a similar approach [41]. In order to evaluate the completeness of his tool, twelve REA class diagrams where redesigned with the REA-DSL. These models were provided by McCarthy who used them in his classes to teach the REA ontology and are based on real companies in the United States. The models vary in complexity and structure and since they are used for teaching purposes they concentrate on different concepts of REA. This was one of the main reasons for using McCarthy's models instead of sticking to smaller, simpler models that can be found in the internet or creating new ones from the scratch. Utilizing them guarantees that all concepts of the REA ontology are included and provides a solid basis for further investigations. Therefore taking these models as foundation for the REA-DSL's evaluation made perfect sense.

Since an existing REA-DSL business model is a mandatory requirement for the artifact that is developed in the context of this thesis, we will use the same models for evaluation purposes as Mayrhofer [41]. As mentioned before this ensures that all relevant concepts are covered. Besides that, we can reuse the models that already have been remodeled with the REA-DSL. Based on these models the functionality and completeness of (i) the insert statement generation for the business models and (ii) the user interface generation prototype will be evaluated.

The twelve business models that will be used are described below. The REA concepts that are used in the corresponding REA-DSL models are also briefly stated.

- **University Slum Lords:** acquisition of resources (value chain consisting of one leasing exchange activity), inside- and outside agents, bulk resource, no hierarchies on agents and resources, only object properties defined for agents and resources.
- **Boston Bottle:** acquisition and revenue cycle (value chain consisting of purchase- and sale exchange activity), inside- and outside agents with specified hierarchy, bulk resources, resourcetypeseries affected in stockflows.
- **Marilyn Monroe Makeovers:** acquisition and revenue cycle (value chain consisting of purchase- and consultation exchange activity), inside- and outside agents, bulk resources, no hierarchies on agents and resources, resourcetypeseries in stockflow, agentseries in participation.
- **Native Alaskan Aircraft Expeditions:** value chain consisting of one expedition exchange activity, inside- and outside agents with specified hierarchy, bulk resources with specified hierarchy, several decrement eventtypes (one decrement eventtypeseries), resource- and resourcetypeseries affected in stockflows, participating agentseries, defined event- and commitment properties for events.
- **Vivian's fashion Factory I:** value chain consisting of one purchase exchange activity, inside- and outside agents, bulk resources, no hierarchies on agents and resources, event properties for event, defined commitments.
- **Brian's Gemini Music:** value chain consisting of one musicjob exchange activity, inside- and outside agents, bulk resource, no hierarchies on agents and resources, several decrement eventtypes (one decrement eventtypeseries), resourcetypeseries affected in stockflows, participating agenttypeseries, defined commitments.
- **Flint X-Ray:** value chain consisting of one officevisit exchange activity, inside- and outside agents, bulk resource, no hierarchies on agents and resources, several eventtypeseries, participating agenttypeseries, defined commitments.
- **South Shore Petroleum:** acquisition and revenue (value chain consisting of refineryjob- and customershipment activity), inside- and outside agents, bulk resources with specified hierarchy, eventtypeseries, resourcetypeseries in stockflow, defined commitments.
- **Western Michigan Office Furniture:** using raw material to manufacture items (value chain consisting of production conversion activity), inside agents, bulk resources, no hierarchies on agents and resources, eventtypeseries, participating agenttypeseries, resourcetypeseries in stockflow, defined commitments.
- **Michigan Medical Equipment:** value chain with one manufacturing conversion activity, hierarchy on inside agents, resourcetypeseries in stockflow, participating agenttypeseries, defined commitments.

- **Nantasket Basket, Gasket, and Casket:** value chain with purchase exchange and production conversion activity, outside- and inside agents with hierarchy, bulk resources, several decrement eventtypeseries, participating agenttypeseries, defined commitments.
- **Vivian's fashion Factory II:** acquisition, manufacturing, and revenue cycle (value chain consisting of purchase/sale exchange activity and production conversion activity), bulk resources with hierarchy, several decrement eventtypeseries, resourcetypeseries affected in stockflows, defined commitments.

8.2 Evaluation of the Database Mappings

In order to evaluate the REA DB insert statements that are generated for REA-DSL business models, the completeness of the SQL script as well as its functionality has to be taken into account. Therefore we define ten indicators that facilitate the decision if the demanded requirements for the scripts are met or not. These indicators are explained in the following.

1. **Agents:** all existing agents that are defined in the agent view of the REA-DSL business model also occur in the SQL script. The generated insert statements for agents contain the correct values for `IsExternal` and `ParentAgenttype_Id`, `Name`, and `Id` (cf. Figure 6.2 on page 62).
2. **Resources:** resources that are defined in the resource view of the REA-DSL business model occur in the generated SQL script. Resource insert statements also contain the correct values for `Id`, `ParentResourceType_Id`, `Name`, `IsBulk`, `IsIdentifiable`, and `UnitOfMeasure` (cf. Chapter 6.1 on page 60).
3. **Dualities:** each activity in the value chain view of the REA-DSL business model occurs in the generated script as insert statement for the dualitytype table. Values for `Id`, `Name`, and `IsConversion` are generated correctly.
4. **Events:** all events or eventtypes that are defined in the planning view of the REA-DSL model occur in the SQL script. Values for `Id`, `ParentEventType_Id`, `Name`, `IsIncrement`, `IsResourceUsed`, and `IsSeries` are defined the right way. Furthermore, each event is related to its corresponding duality as entry in the Dualitytype_Allows_-Eventtype table with appropriate values for `DualityType_Id` and `EventType_Id` (cf. Figure 6.3 on page 64).
5. **Stockflows:** for all stockflows of the events in the planning view, inserts for the Eventtype-Stockflow table are generated. Events are correctly related to the resource that is affected in the stockflow by `EventType_Id` and `ResourceType_Id` values. `IsSeries` and `IsIdentifiable` values are also generated right (cf. Figure 6.3 on page 64).
6. **Participations:** all participation relationships occur in the SQL script with correct values for `EventType_Id`, `AgentType_Id`, `IsSeries` and `IsIdentifiable` (cf. Figure 6.3 on page 64).

7. **Commitments and Contracts:** if commitments are included in the planning view of the REA-DSL model, an insert statement for the Contracttype table is generated. Values for `Id`, `Name`, `IsSchedule`, and the legally committing agents (`IncrementAgentType` and `DecrementAgentType`) are specified correctly. The contract is related to the duality it refers to as entry in the ContractType_Allows_DualityType table with proper values for `ContractType_Id` and `DualityType_Id` (cf. Figure 6.3 on page 64).

8. **Attributes:** all additional attributes that are defined for agents, resources, events, stockflows, and participations occur in the SQL script with values for `Id`, `Name`, and `Datatype`. They are appropriately related to their corresponding entities by insert statements for the AgentType_Has_AdditionalAttribute, ResourceType_Has_AdditionalAttribute, EventType_Has_AdditionalAttribute, EventTypeParticipation_Has_AdditionalAttribute, or EventTypeStockflow_Has_AdditionalAttribute table with respective values (cf. Figure 6.1, 6.2, and 6.4).

9. **Script Functionality:** the generated SQL script executes flawlessly and inserts all data to the corresponding tables of the REA DB.

10. **Inserted records:** the number of data records that are inserted in the database.

	Agents	**Resources**	**Dualities**	**Events**	**Stockflows**	**Participations**	**Commitments / Contracts**	**Attributes**	**Script Functionality**	**Inserted records**
University Slum Lords	✓	✓	✓	✓	✓	✓	–	✓	✓	28
Boston Bottle	✓	✓	✓	✓	✓	✓	–	–	✓	30
Marilyn Monroe Makeovers	✓	✓	✓	✓	✓	✓	–	–	✓	27
Native Alaskan Aircraft Expeditions	✓	✓	✓	✓	✓	✓	–	✓	✓	31
Vivian's fashion Factory I	✓	✓	✓	✓	✓	✓	✓	✓	✓	20
Brian's Gemini Music	✓	✓	✓	✓	✓	✓	✓	–	✓	23
Flint X-Ray	✓	✓	✓	✓	✓	✓	✓	–	✓	28
South Shore Petroleum	✓	✓	✓	✓	✓	✓	✓	–	✓	35
Western Michigan Office Furniture	✓	✓	✓	✓	✓	✓	✓	–	✓	14
Michigan Medical Equipment	✓	✓	✓	✓	✓	✓	✓	–	✓	24
Nantasket Basket, Gasket, and Casket	✓	✓	✓	✓	✓	✓	✓	–	✓	37
Vivian's fashion Factory II	✓	✓	✓	✓	✓	✓	✓	–	✓	50

Table 8.1: Evaluation of REA DB Inserts for twelve different Business Models

Agents, resources, dualities, events, stockflows, particiaptions, commitments/contracts, and attributes are indicators for the completeness of the generated SQL scripts. The functionality is tested by running the script and populating the REA DB. In order to compare the complexity of REA-DSL models that were inserted in the database, the number of inserted records is also listed.

The evaluation results for the twelve business models are represented in Table 8.1 on page 86. A checkmark (✓) in a cell points out that an indicator was tested successfully and no errors occurred. If a test failed the cell contains a ✗. The hyphen (–) depicts that testing an indicator was omitted since the corresponding elements are not represented in the REA-DSL model. As indicated by the number of inserted records in the last column, the twelfth REA-DSL business model (Vivian's fashion Factory II) is the most complex one. This is not surprising since this model has the largest number of activities which leads to an increased number of generated insert statements. Only a few models define additional attributes on elements. Nevertheless it can be concluded that properties are inserted correctly as additional attributes if they exist in the REA-DSL business model.

	Dualitystatus administration	**Agent administration**	**Resource administration**	**Dualityselection**	**Eventcreation**	**Participationcreation**	**Stockflowcreation**	**Dualityvalidation**	**Saving duality**
University Slum Lords	✓	✓	✓	✓	✓	✓	✓	✓	✓
Boston Bottle	✓	✓	✓	✓	✓	✓	✓	✓	✓
Marilyn Monroe Makeovers	✓	✓	✓	✓	✓	✓	✓	✓	✓
Native Alaskan Aircraft Expeditions	✓	✓	✓	✓	✓	✓	✓	✓	✓
Vivian's fashion Factory I	✓	✓	✓	✓	✓	✓	✓	✓	✓
Brian's Gemini Music	✓	✓	✓	✓	✓	✓	✓	✓	✓
Flint X-Ray	✓	✓	✓	✓	✓	✓	✓	✓	✓
South Shore Petroleum	✓	✓	✓	✓	✓	✓	✓	✓	✓
Western Michigan Office Furniture	✓	✓	✓	✓	✓	✓	✓	✓	✓
Michigan Medical Equipment	✓	✓	✓	✓	✓	✓	✓	✓	✓
Nantasket Basket, Gasket, and Casket	✓	✓	✓	✓	✓	✓	✓	✓	✓
Vivian's fashion Factory II	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 8.2: Evaluation of the UI-Generation Prototype

8.3 Evaluation of the UI-Generation Prototype

The UI-Generation prototype was developed to prove that the business models that are saved in the REA DB can act as foundation for the automatically creation of user interfaces. These interfaces should not only reflect all information that was specified in the models, but also enable users to record actual business cases and save them in the generic part of the REA DB. The detailed scope and intended functionality of the UI-Generation prototype was already stated in Chapter 7.3. Similar to the evaluation of SQL inserts, we define indicators that are derived from these preceding thoughts.

1. **Administration of dualitystatuses:** the input for the statuscode is validated and informs the user if unsupported formats were specified. Saving a dualitystatus is only possible if correct values are specified. Apart from saving, dualitystatuses can be updated and deleted.
2. **Administration of agents:** agents can be saved, updated, and deleted without errors. When saving or updating, only agenttypes that are already persisted in the REA DB can be selected. When object- or type properties are defined for agents in the agentview, all necessary forms for specifying the concrete values are generated. User inputs of fixed- and additional attribute values are validated accordingly. User inputs that are not satisfying can not be saved.
3. **Administration of resources:** saving, updating, and deleting existing resources in the REA DB works correctly. Only applicable resourcetypes can be selected. Additional properties are recognized and forms for value definitions are properly created.
4. **Dualityselection:** business cases can only be created for activities that occur in the value chain of stored business models. The options that can be selected by the user reflect the persisted dualitytypes in the REA DB.
5. **Eventcreation:** depending on the selected dualitytype, the user interface for the events is created accordingly. For each event that was defined in the underlying business models operational- or planning view, a tab is created. Furthermore, listboxes for the fixed event attributes (provide- and receive agent) are generated and correctly populated (e.g. only inside agents can receive resources during an increment event). For start- and end date, datepickers are generated to assist the user. If an event has additional properties defined, forms are properly created to enable the definition of additional attribute values.
6. **Participationcreation:** for each participation that was specified in the planning- or operational view of the chosen dualitytype, a tab is created. The prototype only allows the selection of agents that possess the same type (or a subtype) that was defined in the business model. If additional properties exist, forms are created that allow the specification of values. Participation(type)series are recognized and allow the user to create additional participations of the corresponding type.
7. **Stockflowcreation:** similar to participations, the prototype recognizes defined stockflows and creates user interfaces for their specification. The selection of resources is correctly

restricted based on the business model data and forms for all value definitions of additional properties are generated. Furthermore, stockflow(type)series are recognized.

8. **Dualityvalidation:** when trying to save a new duality (with corresponding events, participations, stockflows, and attributes), all user inputs are validated. If values were not properly defined, the prototype informs the user and ensures that nothing is saved in the REA DB.
9. **Saving a duality:** when the user inputs are satisfying, all records are correctly persisted in the REA DB.

Table 8.2 on page 87 lists the results of the UI-Generation validations. The creation of user interfaces worked as intended for all twelve models under investigation. However, it has to be remarked that no commitments are supported by the prototype.

CHAPTER 9

Conclusion and Future Work

9.1 Summary

Customizing ERP systems to specific business demands can be a cumbersome and error-prone task. If not done properly, high follow-up costs can occur. However, customizing is a mandatory step since current ERP systems are often purchased as standard software that is not capable of satisfying all enterprise needs without further adaptations. Furthermore, business requirements can not be regarded as fixed characteristics. When the market changes over time it is critical for the success of an enterprise that the used application systems are capable of adjusting to the changed needs. Current ERP Systems are often not flexible enough to do so. Changed business needs often lead to drastic changes in the data structure and the code, which is inefficient and can cause inconsistencies. Gronau [24] names several techniques that can be used to increase the adaptability of ERP systems and recommends the integration of *business process modeling* for system creation or adaptation. However, the usage of *business modeling languages* seems to be more promising to specify business needs. Since not all tasks have to be specified in detail, the effort for model creation is reduced.

Thus, in this thesis, we illustrated an approach for *business model driven ERP customization*. As business modeling language, the REA ontology was used that has its basis in economic theory and allows the definition of resources, events, and agents. Additionally commitments and policies can be specified which are both important concepts in ERP systems. To simplify the creation of REA business models we used the REA-DSL, a domain specific language capable of defining REA models in an easy and unambiguous way. REA-DSL models are semantically equivalent to the REA class-like models, but grant a clear graphical representation for all REA concepts. Therefore REA-DSL models can be easier understood by domain experts. Furthermore, a tool supports the creation of such models which significantly decreases the model creation effort.

As proposed in the REAlist project, one single database (REA DB) was used to persist business model- as well as business case data. In order to save the REA-DSL business models, a mapping was defined that automatically creates SQL insert scripts and can be executed to populate the REA DB. This data is then used for the definition of actual business cases. To prove that the REA DB can be used as basis for an automatically generation of user interfaces (and therefore reduce the needed customization efforts), a prototype was implemented. Its functionality encompasses the administration of entities that are needed to conduct business cases (dualitystatuses, resources, and agents). Furthermore, the creation of forms for duality-, event-, stockflow-, and participation specification is supported. Users are also able to store the defined business case data in the REA DB.

The main research question that was stated in the beginning of this thesis (“**how can REA-DSL business models be mapped to the REA DB?** “), was answered by the mapping that was defined between REA-DSL models and the database. The technical feasibility of that approach was proven by the realized transformation with the T4 Text Templating Engine. The evaluation results of the generated scripts and the developed UI-Generation prototype is discussed in the next section.

9.2 Evaluation Results

The evaluation of the created artifacts was done with two aspects in mind: (i) the completeness and (ii) the functionality of the generated SQL scripts and the UI-Generation prototype. During development, the evaluation happened in several cycles where the artifacts were tested and, if the results were not satisfying, properly adapted. Based on twelve different business models that include all possible concepts of the REA-DSL and differ in complexity and structure, we defined indicators which facilitate the decision if the generated artifacts satisfy all requirements or not. For the SQL scripts these indicators mainly encompass the completeness of the generated statements (e.g., all resources, events, and agents that are specified in the REA-DSL business model also occur in the scripts). For each REA-DSL business model the insert script was generated and tested based on the predefined indicators. The results of this tests look promising (cf. Table 8.1 on page 86). The generated SQL scripts include all concepts of the REA-DSL models. Commitments and additional properties are only recognized if the business model actually includes those concepts. Moreover, executing the scripts works and actually populates the REA DB with business model data which proves the functionality of the artifact. The evaluation of the UI-Generation prototype was based on the same principle. Again, indicators were defined that helped us to test the completeness as well as the functionality. These indicators are based on the intended scope of the UI-Generation prototype that was outlined in Chapter 7.3. Table 8.2 on page 87 shows the test results for the twelve business models. As illustrated, the outcome of the prototype evaluation for each indicator is satisfying. Dualities can be defined for each activity that occurs in the value chain view of the underlying business model. The number of needed events is recognized and the respective sections are included in the user interface. All participations and stockflows that were defined in the planning view models are included in the generated masks. Furthermore, the user interface can be used to define actual business cases. The entered data is stored in the respective tables of the REA DB as intended which proves the functionality of the UI-Generation prototype. The positive test results come by no surprise since the indicators are based on the predefined scope of the prototype. Therefore, even if the evaluation for distinct indicators failed at some point, the prototype was adapted until it fulfilled all the specified needs. Based on the evaluation results of the generated SQL scripts and the UI-Generation prototype, we can conclude that using our approach REA-DSL business models can be mapped to the REA DB. Moreover, we proved that it is possible to base the generation of user interfaces on the persisted business model data and use these masks to save actual business case data in the database.

9.3 Limitations and Future Work

Supporting commitments. The UI-Generation prototype currently does not support commitments that were defined in REA-DSL models. The next step therefore should be to realize the generation of masks for these concepts. This encompasses a section for specifying contracts with an existing `provide-` and `receive agent` that legally commits to it. The `date` when the contract is concluded also is persisted in the database and therefore has to be specified by the user. Similar to dualities, contracts have a certain `status`. Therefore the entity administration part of the UI-Generation prototype needs to be extended to support the creation, update, and removal of contract statuses. Furthermore, masks for the definition of commitments that are fulfilled by the actual events have to be generated. The structure of that user interface basically looks similar to the masks that were created for the events at the operational level. Additional attribute values that can be defined in the user interfaces of the commitment sections belong to the commitment- or reserve properties that are specified for events, participations, and stockflows.

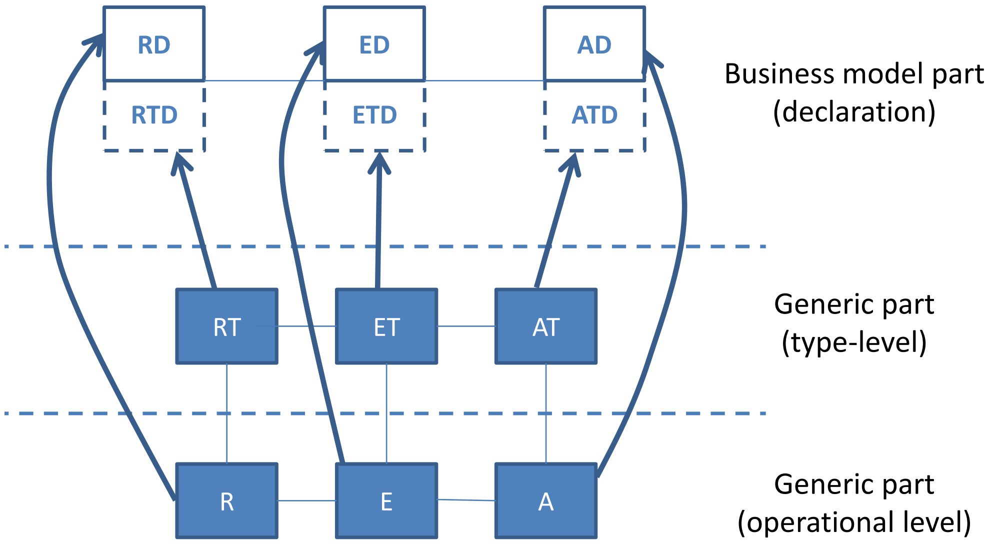


Figure 9.1: Extending the REA DB by a Declaration Layer

Adapting the REA DB data structure to include a dedicated declaration part for business models. During multiple meetings with Dieter Mayrhofer, the data structure of the REA DB was discussed and revised several times. In its current form, no distinct tables for the business model concepts are included. As illustrated in Chapter 5 and further discussed in Chapter 6, type tables are used to store the business model data. Therefore, it is not clear if an entry in such a table represents a model element or an actual type (as proposed by Geerts et al. [15, 17]). However, type properties should only be defined for type entities. An example is the startingwage of an employee (cf. Figure 4.2 on page 41). In its current form, it is not clear when this property should be declared. We therefore allow the user to specify values for the startingwage for each concrete agent with agenttype employee (or one of its subtypes). For further extensions of the REA DB, a third layer should be introduced that distinguishes type entities from business model elements. This idea is visualized in Figure 9.1. The advantage of doing so is that all the REA elements are clearly separated from the business model data in the database. Tables for events, resources, and agents exist as well as tables for their type entities. This allows us to define actual values for the properties of types. For example, the value of the intended startingwage for an employee type could be specified and evaluated against the actual wage of a concrete agent with the corresponding type. All REA-DSL elements are persisted in tables of the declaration layer.

Including all parts of the generic REA DB in the UI generation. When developing the UI-Generation prototype, only the necessary parts of the generic REA DB were used. As illustrated in Appendix A, the database tables are strongly related with each other. E.g., resources also possess a `taxclass` that is represented as entry in the Tax table. To demonstrate that the business model data can be used for adapted user interface creation, this relation was omitted. Of course, in further extensions users should be able to specify taxclasses as well. Therefore an administration section for saving, updating, and deleting taxes has to be generated similar to dualitystatuses, agents, and resources.

Optimization of the generated user interfaces. The generated masks to save business case data are just prototypical. For future improvements, the look and feel of the user interfaces has to be improved to guarantee a satisfying user experience. The Google Web Toolkit[1] that was used to create the prototype enables software developers to extensively use Cascading Style Sheets (CSS) to apply styles on forms.

Multitenancy support. The medium-term goal of the REAlist project is to support multitenancy for the ERP system. Therefore business models that are saved by tenants need to be distinguishable somehow. One possibility is to add an identification column in the DualityType table that specifies which activities belong to certain users. The duality selection part of the generated interfaces could then only display business model data that was defined for a specific company to the logged in user.

[1] `http://www.gwtproject.org/`

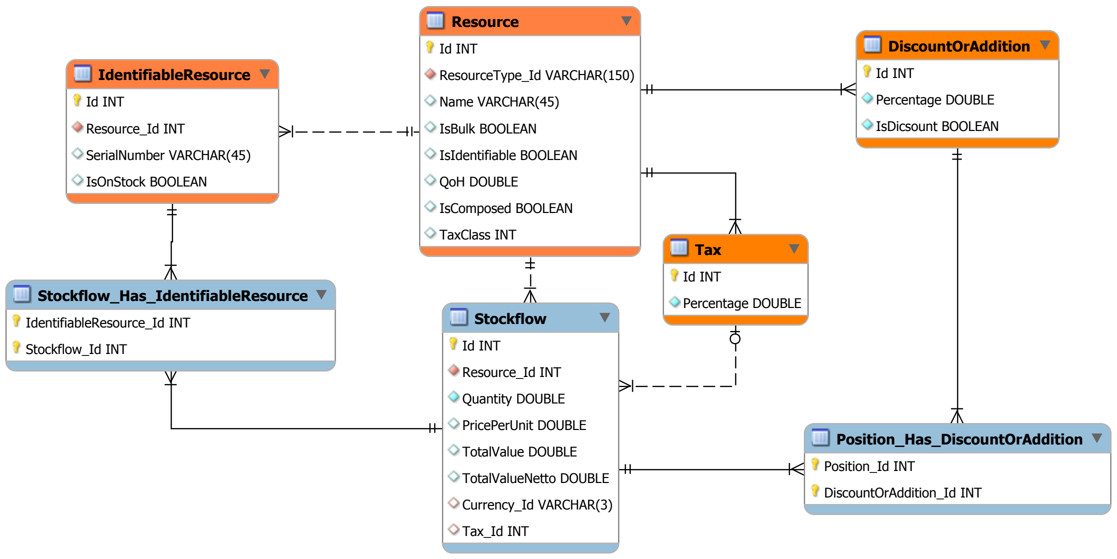
APPENDIX A

Subparts of the REA DB

This section illustrates all parts of the REA DB that were not already extensively discussed in the course of this thesis. To facility a better understanding, the tables are structured in subparts that depict the needed functionality of an ERP system.

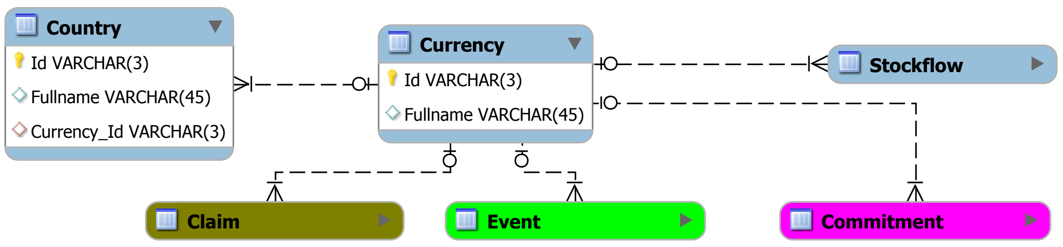
A.1 Resources

Subpart of the REA DB that contains tables for storing resources, identifiable resources, and stockflows.



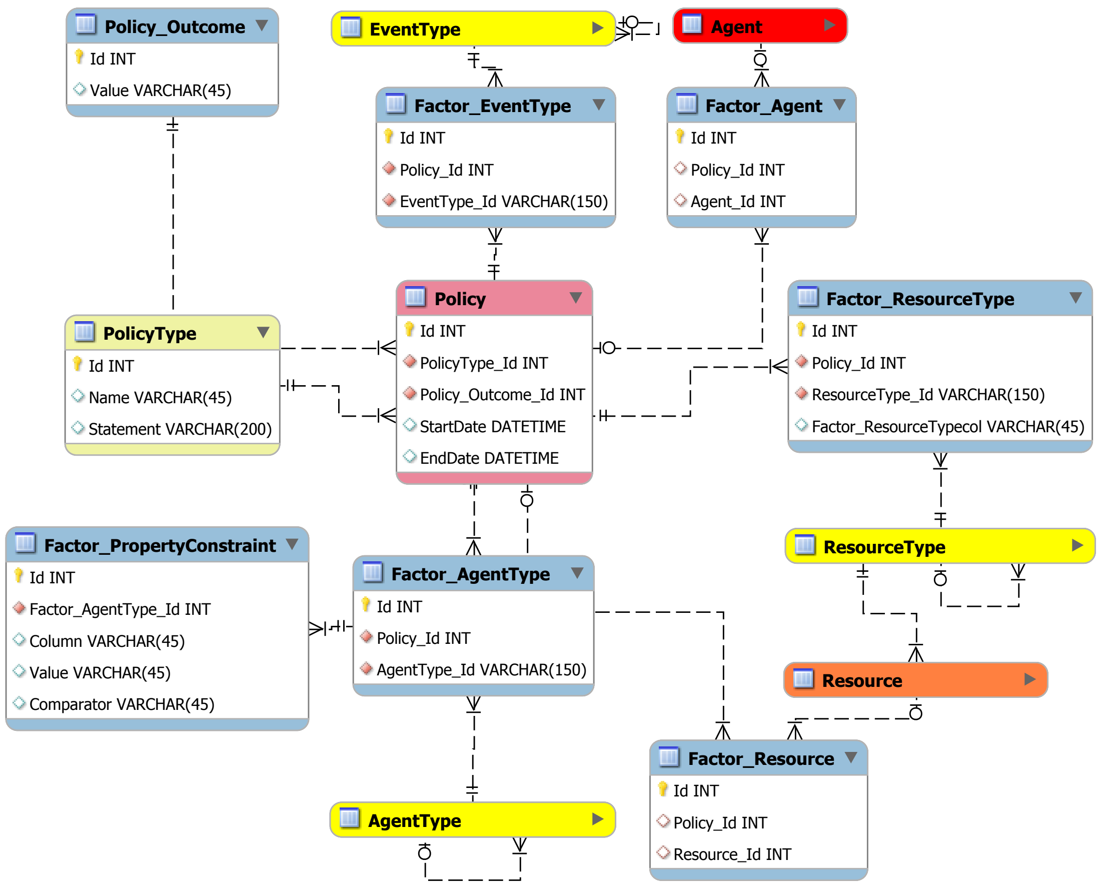
A.2 Currency and Country

Subpart of the REA DB containing tables for currency and country specifications.



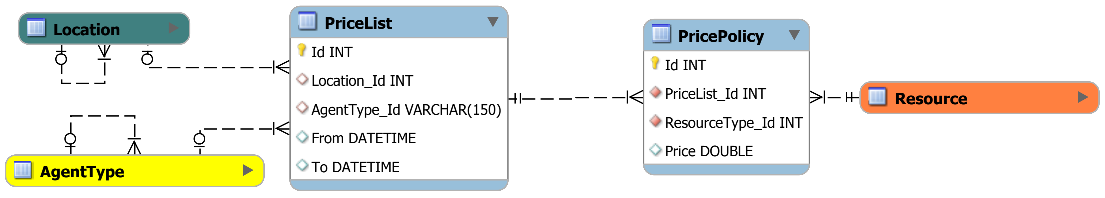
A.3 Policy

Subpart of the REA DB that allows the definition of very simple policies.



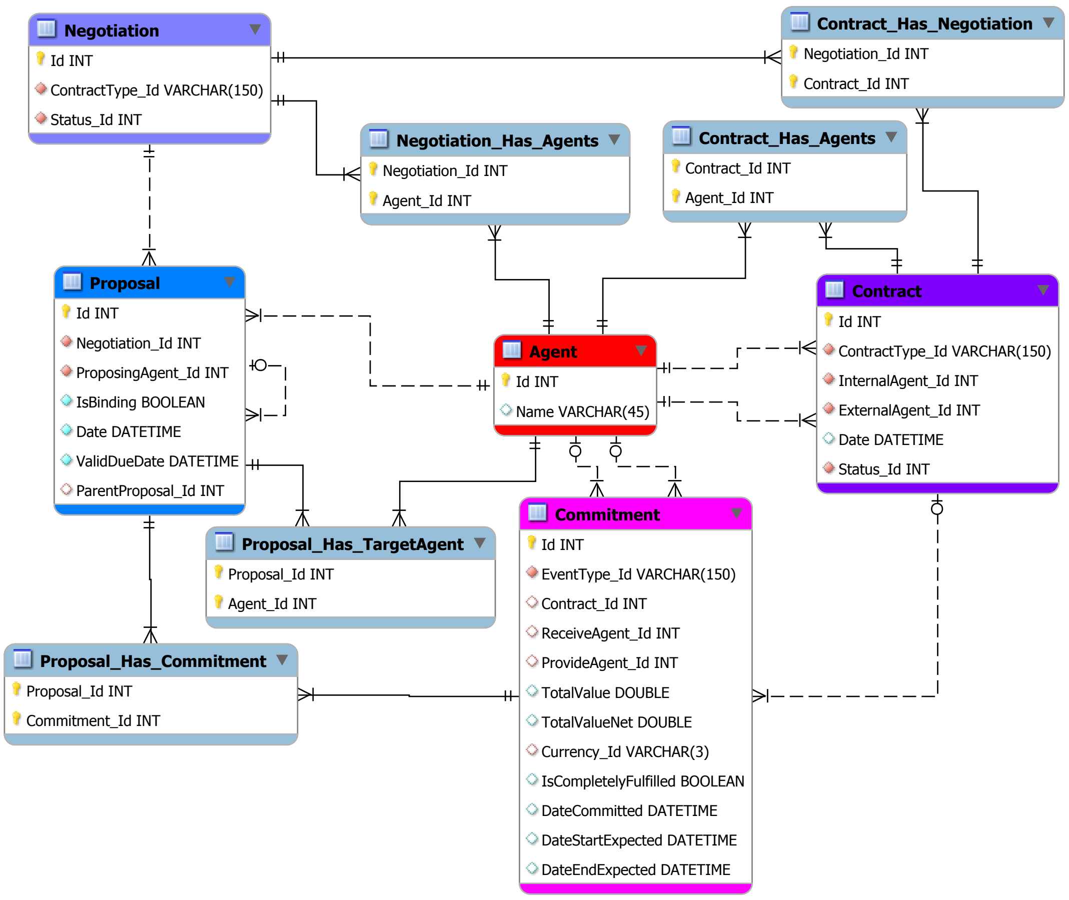
A.4 Pricelist

Subpart of the REA DB that permits the definition of individual prices for agenttypes and locations during specific timeframes.



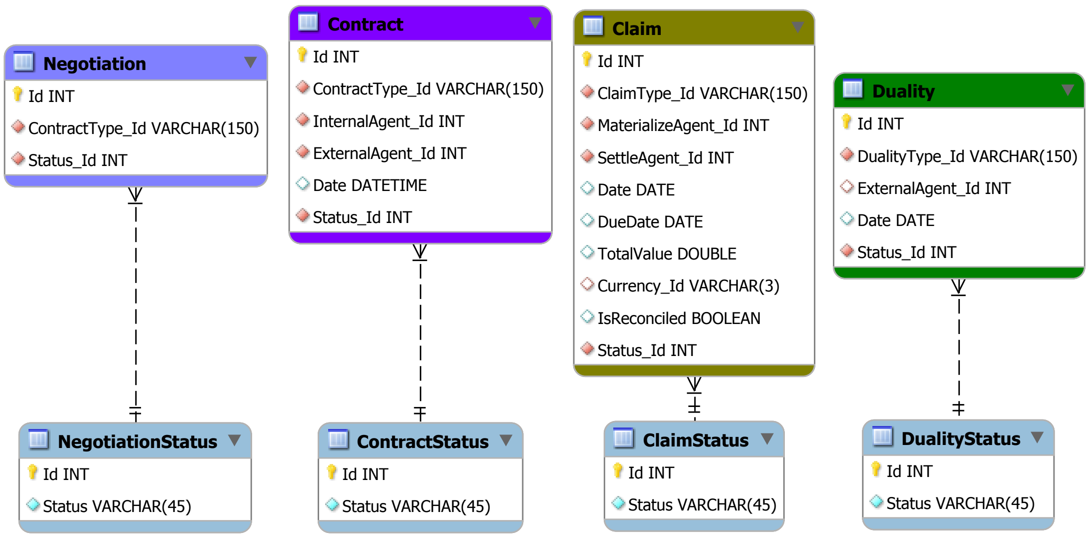
A.5 Negotiation

REA DB part that contains tables that allow to keep track of negotiations. The course of a negotiation can be reproduced.



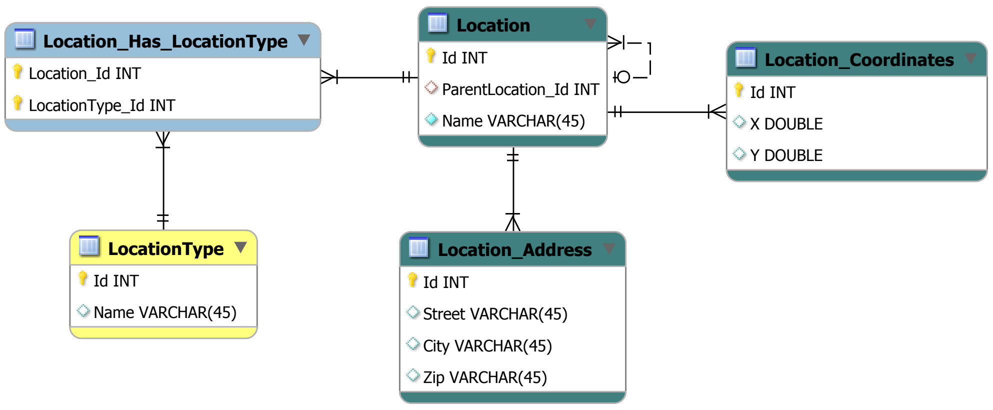
A.6 Status

Part of the REA DB specifying statuses of negotiation, claim, duality, or contract.



A.7 Location

Subpart of the REA DB that contains tables for locations that can be defined for various REA concepts.



List of Figures

List of Tables

Bibliography

[1] B. Andersson, M. Bergholz, A. Edirisuriya, T. Ilayperuma, P. Johannesson, J. Gordijn, B. Grégoire, M. Schmitt, E. Dubois, S. Abels, A. Hahn, B. Wangler, and H. Weigand. Towards a Reference Ontology for Business Models. In *Proceedings of the 25th International Conference on Conceptual Modeling*, pages 482–496. Springer, 2006.

[2] H. J. Appelrath and J. Ritter. *R/3 Einführung*. Springer Verlag, 1. edition, 2000.

[3] N. H. Bancroft, A. Sprengel, and H Seip. *Implementing Sap R/3 : How to Introduce a Large System into a Large Organization*. Manning Publications Co., 2. edition, 1998.

[4] P. P. Chen. The Entity-Relationship Model - Toward a Unified View of Data. *ACM Transcactions on Database Systems*, 1(1):9–36, 1976.

[5] L. Columbus. 2013 ERP Market Share Update: SAP Solidifies Market Leadership. `http://www.forbes.com/sites/louiscolumbus/2013/05/12/2013-erp-market-share-update-sap-solidifies-market-leadership/`, 2013. Online, last accessed: 13.01.2014.

[6] L. Columbus. SaaS Adoption Accelerates, Goes Global in the Enterprise. `http://www.forbes.com/sites/louiscolumbus/2012/10/31/saas-adoption-accelerates-goes-global-in-the-enterprise/`, 2013. Online, last accessed: 13.01.2014.

[7] S. Cook, J. Gareth, K. Stuart, and W. Alan Cameron. *Domain-Specific Development with Visual Studio DSL Tools*. Addisom-Wesley, 1. edition, 2007.

[8] M. J. Cotteleer and E. Bendoli. Order Lead-time Improvement Following Enterprise-IT Implementation: An Empirical Study. *MIS Quarterlyt*, 30(3):643–660, 2006.

[9] B. Curtis, M. I. Kellner, and J. Over. Process Modeling. *Communications of the ACM*, 35(9):75–90, 1992.

[10] T. H. Davenport. *Process Innovation: Reengineering Work Through Information Technology*. Harvard Business School Press, 1. edition, 1993.

[11] V. De Casto and E. Marcos. Towards a Service-oriented Approach to the Alignment of Business Processes with IT-Systems: From the Business Model to a Web Service Composition Model. *International Journal of Cooperative Information Systems*, 18(2):225–260, 2009.

[12] M. Fowler. *Patterns of Enterprise Application Architecture*. Addison Wesley, 1. edition, 2002.

[13] G. Geerts and W. E. McCarthy. Augmented Intensional Reasoning in Knowledge-Based Accounting Systems. *Journal of Information Systems*, 14(1):127–150, 1993.

[14] G. Geerts and W. E. McCarthy. Modeling Business Enterprises as Value-Added Process Hierarchies with Resource-Event-Agent Object Templates. In *Proceedings of the Workshop on Business Object Design and Implementation in conjunction with the Conference on Object-Oriented Programming Systems, Languages & Applications (OOPSLA 1995), Atlanta, Georgia*, pages 94–113. Springer-Verlag, 1997.

[15] G. Geerts and W. E. McCarthy. The Ontological Foundation of REA Enterprise Information Systems. `https://www.msu.edu/~mccarth4/Alabama.doc`, 2000. Online, last accessed: 13.01.2014.

[16] G. Geerts and W. E. McCarthy. Using Object Templates From The REA Accounting Model To Engineer Business Processes And Tasks. *The Review of Business Information Systems*, 5(4):89–108, 2001.

[17] G. Geerts and W. E. McCarthy. Type-Level Specifications in REA Enterprise Information Systems. `https://www.msu.edu/user/mccarth4/UTS-seminar/Type%20paper%20final%20submission.doc`, 2004. Online, last accessed: 13.01.2014.

[18] G. Geerts and W. E. McCarthy. Policy-Level Specifications in REA Enterprise Information Systems. *Journal of Infrmation Systems*, 20(2):37–63, 2006.

[19] C. Gerth. *Business Process Models. Change Management*. Springer, 1. edition, 1985.

[20] J. Gordijn and H. Akkermans. e3-value: Design and Evaluation of e-Businss Models. *IEEE Intelligent Systems*, 16(4):11–17, 2001.

[21] J. Gordijn and H. Akkermans. Value Based Engineering: Exploring Innovative e-Commerce Ideas. *Requirements Engineering Journal*, 8(2):114–134, 2003.

[22] J. Gordijn, H. Akkermans, and H. van Vliet. Business Modeling is not Process Modeling. In *Proceedings of the Workshop on Conceptual Modeling for E-Business and the Web (ECOMO 2000))*, pages 40–51. Springer, 2000.

[23] P. A. Grammer. *Der ERP-Kompass: ERP-Projekte zum Erfolg führen*. mitp-Verlag, 1. edition, 2011.

[24] N. Gronau. *Enterprise Resource Planning*. oldenbourg.verlag, 2. edition, 2010.

[25] N. Gronau and M. Lindemann. *Einführung in das Informationsmanagement*. gito.verlag, 1. edition, 2010.

[26] Object Management Group. Meta Object Facility (MOF) Core Specification, Version 2.4. `http://www.omg.org/spec/MOF/2.4.1/PDF/`, 2011. Online, last accessed: 13.01.2014.

[27] Object Management Group. Business Process Model and Notation (BPMN), Version 2.0. `http://www.omg.org/spec/BPMN/2.0/PDF/`, 2013. Online, last accessed: 13.01.2014.

[28] T. Gruber. Towards Principles for the Design of Ontologies Used for Knowledge Sharing. In *Formal Ontology in Conceptual Analysis and Knowledge Representation, Deventer, The Netherlands*. Kluwer Academic Publishers, 1993.

[29] K. B. Hendricks, V. R. Singhal, and J. K. Startman. The impact of enterprise systems on corporate performance: A study of ERP, SCM, and CRM system implementations. *Journal of Operations Management*, 25(1):65–82, 2007.

[30] M. Hesseler. Customizing von ERP-Systemen. Rollenbasierte Konzepte bieten neue Möglichkeiten für individualle Anpassungen. *Controlling & Managemen Review*, 53(3):48–55, 2009.

[31] A. R. Hevner, S. T. March, and J. Park. Design Science in Information Systems Research. *MIS Quarterly*, 28(1):75–105, 2004.

[32] P. Hruby. *Model-Driven Design Using Business Patterns*. Springer, 1. edition, 2006.

[33] P. Hruby. REA (Resources, Events, Agents) - Presentation at Vienna University of Technology. `http://www.ec.tuwien.ac.at/files/etrends2008/PavelHruby.pdf`, 2008. Online, last accessed: 13.01.2014.

[34] C. Huemer and D. Mayrhofer. REAlist - Mandantenfähiges ERP System in der Cloud: Ein modellgetriebener Ansatz auf Basis der Resource-Event-Agent (REA) Ontologie, 2013. FFG BRIDGE Projektantrag.

[35] Y. Ijiri. *Theory of Accounting Measurement*. American Accounting Association, 1. edition, 1975.

[36] R. Jacobs and T. Jr. Weston. Enterprise Resource Plannning (ERP) - A brief history. *Journal of Operations Management*, 25(2):357–363, 2007.

[37] R. S. Kaplan and D. R. Norton. The Balanced Scorecard: Measures That Drive Perfermance. *Harvard Business Review*, 83(7/8):172–180, 2005.

[38] V. Kulkarni, S. Reddy, and A. Rajbhoj. Scaling up model driven engineering-experience and lessons learnt. In *Proceedings of the 13th International Conference on Model DrivenEengineering Languages and Systems: Part II (Berlin, Heidelberg, 2010), MODELS '10*, pages 331–345. Springer-Verlag, 2010.

[39] P. Loos and T. Theling. Marktübersicht zu ERP-Literatur. `http://www.econbiz.de/archiv/mz/umz/winformatik/marktuebersicht_erp-literatur.pdf`, 2003. Online, last accessed: 13.01.2014.

[40] J. Magretta. Why Business Models Matter. *Harvard Business Review*, 80(5):86–92, 2002.

[41] D. Mayrhofer. *REA-DSL: Business Model Driven Data Engineering*. PhD thesis, Vienna University of Technology, 2012.

[42] D. Mayrhofer and C. Huemer. Business-Model-Driven Data Engineering Using the REA-DSL. In *Proceedings of the 6th International Workshop on Value Modeling and Business Ontology (VMBO 2012), Vienna, Austria*, 2012.

[43] D. Mayrhofer and C. Huemer. Extending the REA-DSL by the Planning Layer of the REA Ontology. In *Proceedings of the 7th International Workshop on Business/IT-Alignment and Interoperability (BUSITAL 2012), in conjunction with the 24th International Conference on Advanced Information Systems Engineering (CAiSE 2012), Gdansk, Poland*. Springer, 2012.

[44] D. Mayrhofer and C. Huemer. REA based OnlineERP: FFG Feasibility Study - Endbericht, 2013.

[45] D. Mayrhofer, C. Sonnenberg, B. Hofreiter, and C. Huemer. A Domain Specific Modeling Language for REA. In *Proceedings of the 6th International Workshop on Value Modeling and Business Ontologies (VMBO 2012), Vienna, Austria*, 2012.

[46] A. McAfee. The Impact of Enterprise Information Technology Adoption on Operational Performance: An Empirical Investigation. *Productions and Operations Management*, 11(1):33–53, 2002.

[47] W. E. McCarthy. An Entity-Relationship View of Accounting Models. *The Accounting Review*, 54(4):667–686, 1979.

[48] W. E. McCarthy. Construction and Use of integrated Accounting Systems with Entity-Relationship Modeling. In *Proceedings of the 1st International Conference on the Entity-Relationship Approach to Systems Analysis and Design*, pages 625–637. ACM, 1980.

[49] W. E. McCarthy. The REA Accounting Model: A Generealized Framework for Accounting Systems in a Shared Data Environment. *The Accounting Review*, 52(1):554–578, 1982.

[50] B. McNurlin and R. Sprague. *Information Systemns Management in Practice*. Pearson, 5. edition, 2002.

[51] V. Meister. *Grundlagen betrieblicher Anwendungssysteme*. expert verlag, 1. edition, 2011.

[52] P. Mertens. *Integrierte Informationsverarbeitung 1 - Operative Systeme in der Industrie*. Gabler Verlag, 17. edition, 2009.

[53] H. Mili, G. Bou Jaoude, É Lefebvre, G. Tremblay, and A. Petrenko. Business Process Modeling Languages: Sorting Through the Alphabet Soup. *ACM Computing Surveys*, 43(1):4:1–4:56, 2010.

[54] D. E. O'Learyy. On the relationship between REA and SAP. *International Journal of Accounting Information Systems*, 5(1):65–81, 2004.

[55] A. Osterwalder. *The Business Model Ontology. A Proposition in a Design Science Approach.* PhD thesis, University of Lausanne, 2004.

[56] A. Osterwalder and Y. Pigneur. An ontology for e-business models. In W. L. Currie, editor, *Value Creation from e-Business Models*, pages 65–97. Butterworth-Heinemann, 2004.

[57] A. Osterwalder, Y. Pigneur, and C. L. Tucci. Clarifying Business Models: Origins, Present, and Future of the Concept. *Communications of the Association for Information Systems*, 15(1):1–40, 2005.

[58] M. E. Porter. *Competitive Advantage*. The Free Press, 1. edition, 1985.

[59] M. Rappa. Business Models on the Web. `http://digitalenterprise.org/models/models.html`, 2005. Online, last accessed: 13.01.2014.

[60] K. Rosli, A. Ahmi, and L. Mohamad. Resource-Event-Agent (REA) Modeling in Revenue Information System (RIS) Development: Smart Application for Direct-Selling Dealers and SMEs. *Journal for the Advancement of Science and Arts*, 1(1):43–62, 2009.

[61] T. A. Sedbrook. Modeling the REA Enterprise Ontology with a Domain Specific Language. *Journal of Emerging Technologies in Accounting*, 9(1):47–70, 2012.

[62] J. Smith David, C. L. Dunn, and W. E. McCarty. Enterprise resource planning systems research: The necessity of explicating and examining patterns in symbolic form. In *1st International Workshop on Enterprise Management and Resource Planning System*. Venice, Italy, 1999.

[63] C. Sonnenberg, C. Huemer, B. Hofreiter, D. Mayrhofer, and A. Braccini. The REA-DSL:A Domain Specific Modeling Language for Business Models. In *Proceedings of the 23rd International Conference of Advanced Information Systems Engineering (CAiSE 2011), London, UK*, pages 252–266. Springer-Verlag, 2011.

[64] P. Timmers. Business Models for Electronic Markets. *Journal on Electronic Markets*, 2(2):3–8, 1998.

[65] A. van Deursen, P. Klint, and J. Visser. Domain-Specific Languages. *ACM SIGPLAN*, 35(6):26–36, 2000.

[66] R. Weber. *Technologie von Unternehmenssoftware*. Springer Vieweg, 1. edition, 2012.

[67] S. C. Yu. *The Structure of Accounting Theory*. The university Presses of Florida, 1. edition, 1976.

[68] U. Zdun and M. Strembeck. An Approach for the Systematic Development of Domain-Specific Languages. *Software - Practice and Experience (SPE)*, 39(15):1253–1292, 2009.

[69] C. Zott and R. Amit. Designing your future Business Model: An Activity System Perspective. *Long Range Planning*, 37:216–226, 2010.

[70] C. Zott, R. Amit, and L. Massa. The Business Model: Recent Developments and Future Research. *Journal of Management*, 37(4):1019–1042, 2011.