

REA-DSL: Business Model Driven Data Engineering

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der technischen Wissenschaften

eingereicht von

Dipl.-Ing. Mag. Dieter Mayrhofer

Matrikelnummer 0325946

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: ao.Univ.Prof. Mag. Dr. Christian Huemer

Diese Dissertation haben begutachtet:

(ao.Univ.-Prof. Mag. Dr.
Christian Huemer)

(Prof. Dr. William E. McCarthy)

Wien, 01.06.2012

(Dipl.-Ing. Mag. Dieter
Mayrhofer)

REA-DSL: Business Model Driven Data Engineering

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der technischen Wissenschaften

by

Dipl.-Ing. Mag. Dieter Mayrhofer

Registration Number 0325946

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: ao.Univ.Prof. Mag. Dr. Christian Huemer

The dissertation has been reviewed by:

(ao.Univ.-Prof. Mag. Dr.
Christian Huemer)

(Prof. Dr. William E. McCarthy)

Wien, 01.06.2012

(Dipl.-Ing. Mag. Dieter
Mayrhofer)

Erklärung zur Verfassung der Arbeit

Dipl.-Ing. Mag. Dieter Mayrhofer
Ungerbachstrasse 32, 2860 Kirchsschlag

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

Taking the challenge to pursue a PhD requires a lot of commitment, flexibility, hard work, and willingness to learn. But most importantly, finishing a thesis would be impossible without the help and support of many other smart and inspiring people. These people also made the time as a PhD student a fun and an unforgettable one.

First of all, I would like to thank my advisor and mentor, Christian Huemer. He guided me throughout the last three years of my PhD, spent numerous hours with me discussing my thesis and papers, and helped me with many ideas. I would also like to thank Gerti Kappel, who always took her time to answer my questions and supported me with my thesis.

I am also grateful for my external advisor, Bill McCarthy, who undoubtedly made this thesis possible by creating the foundations – the REA ontology – thirty years ago, when I was actually born. Thank you for all your help, the weekly Skype calls and the great time in Michigan which helped me a lot.

Especially, I am also thankful for all the ducklings of the ENTE project team, Marco Zapletal, Philipp Liegl, Rainer Schuster, Philipp Krenn, Mario Topf, and Michael Pöttler. Besides supporting me in various ways with my thesis, they also made sure that I do not forget and enhance my technical abilities in this awesome startup.

Thanks to all my other colleagues, Robert Engel, Philip Langer, Konrad Wieland, Christian Pichler, Tanja Mayerhofer, Thomas Motal, Manuel Wimmer, and many others helping me finish this thesis and providing valuable input. It was always a fun time with you guys at work as well as after work.

In particular, I would like to thank my family and friends. Throughout the last thirty years, my parents Alfred and Hilda made sure that I have a pleasant life and supported me in any possible way. Together with my brothers and grandparents Grotti, Opa, and Oma, they have always been an essential and inspiring part of my life. Thanks as well to Gerhard and Andrea, who always made sure that I can take a break from work, kick back, and relax.

Most importantly, I am so grateful for my wonderful girlfriend Lisa, who has always been there for me. She makes sure that I feel good when there are hard times and supports me with all her love. It is always fun doing countless great activities with her. Thanks for this unforgettable time.

Abstract

Accounting Information Systems (AIS) are essential for companies not only to record and track what events are happening or what events have happened in the past, they are also one of the most important tools for management to predict the financial future of a firm and take according actions. To enable an AIS to precisely record economic data and apply reasoning on them, it is crucial that the data structure of an AIS is built upon the economic phenomenas of a company's business. Accordingly, it is already important to have a detailed understanding of the company's economic actions at the design time of an AIS.

Like in most software development projects, the dilemma during design time of an IT product is the language barrier between the domain expert, providing vital input for defining the requirements, and the IT professional, designing and developing the IT product. In most cases the domain expert is not capable of understanding IT specific terms, and the IT professional is not capable of completely understanding the specific area of the domain expert. Nevertheless, to successfully complete an IT product, these two groups still need to unambiguously communicate with each other by using a common language.

When designing an AIS, the domain at hand is the accounting/business domain. Thus, a business modeling language describing economic phenomenas of a company can be used as such a common language to define requirements. One powerful business modeling language today is the Resource-Event-Agent ontology (REA). It not only allows describing events of the present and the past, it also allows specification of commitments made for future events. Consequently, it perfectly fits our requirement to capture the economic phenomenas of an AIS. However, REA is somewhat vague in the definition of its concepts and the current representation is merely IT related, which makes it hard to be understood by business experts. Accordingly, REA still cannot be used as a common communication language for the AIS design phase.

Given these limitations, in this thesis we have taken upon the challenge to develop an unambiguous and intuitive graphical domain-specific representation for the REA ontology called the REA-DSL. First, we formalize the REA ontology by providing a REA-DSL meta-model incorporating the REA concepts resources, events, agents, commitments, and types as well as concepts known from database modeling. Subsequently, we create a graphical notation for the REA-DSL using different shapes for different REA concepts. Additionally, to reduce the complexity of the models, we split the REA-DSL into five interlinked views. A serialization format for the REA-DSL is provided by the REA-XML language. Furthermore, we specify a mapping between the conceptual REA-DSL and a database description language. This enables the semi-automatic generation of database structures for an AIS.

The presented REA-DSL serves as an unambiguous and powerful business modeling language which can be used by IT and business experts for faster designing a robust AIS.

Kurzfassung

Rechnungswesen-Informationssysteme (AIS - Accounting Information Systems) sind nicht nur wichtig um aktuelle und vergangene betriebliche Aktivitäten aufzuzeichnen, sie sind auch ein essentielles Werkzeug für das Management einer Firma um Voraussagen über die zukünftige finanzielle Lage zu treffen und entsprechende betriebliche Entscheidungen zu treffen. Damit ein AIS betriebliche Daten strukturiert und automatisch verarbeitbar speichern kann ist es unerlässlich, dass die Datenstruktur des AIS die ökonomischen Gepflogenheiten einer Firma widerspiegelt. Daher ist ein detailliertes Verständnis der betrieblichen Aktivitäten einer Firma während der Entwicklung eines AIS von besonderer Bedeutung.

Ein häufiges Problem bei der Entwicklung von IT Produkten ist die Sprachbarriere zwischen den Domänenexperten – welche die Anforderungen festlegen – und den IT-Experten – welche das IT System implementieren. In vielen Fällen verstehen die Domänenexperten die IT Begriffe nicht und umgekehrt die IT-Experten die Begriffe einer ihnen fremden Domäne nicht. Um jedoch ein Software Produkt erfolgreich zu entwickeln, muss eine gemeinsame eindeutig verständliche Sprache gefunden werden.

Bei der Entwicklung eines AIS ist diese spezifische Domäne die der Betriebswirtschaft und des Rechnungswesens. Die Geschäftsmodellierungssprache Resource-Event-Agent Ontologie (REA), welche die aktuellen und zukünftigen wirtschaftlichen Aktivitäten einer Firma beschreibt, dient als eine solche gemeinsame Sprache zur Anforderungsanalyse eines AIS. Die Hauptprobleme von REA sind jedoch, dass es erstens zurzeit keine ausreichende und daher eindeutige Formalisierung dafür gibt, und zweitens die grafische Repräsentation aus der IT geerbt wurde und daher von Betriebswirtschaftsexperten nur schwer verstanden wird.

Um diese Probleme zu lösen, präsentieren wir in dieser Dissertation eine eindeutige und intuitive graphische domänenspezifische Sprache für die REA Ontologie – die REA-DSL. Als Grundlage definieren wir ein Meta-Model, welches die REA Konzepte Resource, Event, Agent, Commitment und Type sowie deren Beziehungen untereinander formalisiert. Darauf aufbauend spezifizieren wir eine grafische Notation, welche verschiedene Formen für die diversen REA Konzepte appliziert. Um eine Reduktion der Komplexität der REA Modelle zu erreichen, werden die einzelnen Modelle in fünf miteinander verbundene spezifische Ansichten aufgeteilt. Ein Serialisierungsformat der REA-DSL wird durch eine eigene REA-XML Sprache bereitgestellt. Des Weiteren beschreiben wir eine Transformation zwischen unserer REA-DSL und einer Datenbank Definitionssprache für den direkten Einsatz von AIS Datenbanksystemen.

Die REA-DSL ist eine eindeutige und ausdrucksstarke Geschäftsmodellierungssprache, die zusammen von IT- und Betriebswirtschaftsexperten für das beschleunigte Entwickeln eines robusten AIS eingesetzt werden kann.

Contents

1	Introduction	1
1.1	Motivation	2
1.2	Problems and Contributions	5
1.3	Methodological Approach	10
1.4	Structure of this Thesis	13
2	State of the Art	15
2.1	Business Models	16
2.1.1	BMO - Business Model Ontology	17
2.1.2	e ³ value	19
2.1.3	REA	20
2.2	Value Chain Modeling	21
2.3	Domain-Specific Languages	23
2.4	Database Modeling	27
3	REA Overview	31
3.1	REA Core Concepts	32
3.1.1	Vertical Layers	34
3.1.2	Horizontal Layers	35
3.2	REA Representation and Formalization Approaches	39
3.3	REA in Accounting Information Systems	41
3.4	REA Extensions and Integrations	42
3.5	REA Applications and REA Mappings	43
4	REA by Example - Sy's Fish Wholesaler	45
4.1	The Business Idea	45
4.2	REA Class Diagram Representation	46
4.3	REA Ontology Limitations	49
5	REA-DSL	53
5.1	REA Notation Elements	55
5.2	Agent View	59
5.3	Resource View	60

5.4	Operational View	63
5.5	Value Chain View	68
5.6	Planning View	72
5.6.1	Meta-Model and DSL	72
5.6.2	Mapping Between Planning View and Operational View.	77
5.6.3	Policies Contained in the Planning View	77
5.7	Example	79
5.7.1	Agent View	79
5.7.2	Resource View	80
5.7.3	Value Chain View	81
5.7.4	Operational View	81
5.7.5	Planning View	84
6	REA-DB	87
6.1	REA-DSL Extended: Adding Properties and Keys	88
6.2	REA-DSL to Relational Model Mapping	93
6.2.1	Agent View Mapping	93
6.2.2	Resource View Mapping	95
6.2.3	Operational View Mapping	98
6.2.4	Planning View Mapping	101
7	REA-XML	107
7.1	Agent View	109
7.2	Resource View	111
7.3	Operational View	115
7.4	Value Chain View	119
7.5	Planning View	123
8	Evaluation and Tool Support	129
8.1	REA-DSL Tool	129
8.1.1	Tool Creation	129
8.1.1.1	Specification of the REA-DSL Meta-Model	130
8.1.1.2	Interaction with the REA-DSL	132
8.1.1.3	REA-DSL to Relational Model Mapping Implementation	133
8.1.2	Tool Description	133
8.1.2.1	Tool User Interface	133
8.1.2.2	REA-DSL Modeling Steps	136
8.2	Redesigning Existing REA Models	141
8.3	Expert Interviews	144
8.4	Usability Study	144
8.5	Hypotheses Verification	149
9	Conclusion and Future Work	151
9.1	Contributions	153

9.2	Evaluation Summary	155
9.3	Limitations and Future Work	155
A	Usability Study Documents	159
A.1	Teaching Examples	160
A.2	Evaluation Examples	162
A.3	Questionnaire	164
B	OCL Constraints	167
B.1	Operational View OCL Constraints	167
B.2	Planning View OCL Constraints	168
	List of Figures	170
	Nomenclature	172
	Bibliography	175

Introduction

The history of accounting dates back more than 7000 years when farmers in the Middle East kept track of their harvest and animals by simply counting them [FP96]. This way, they could easily calculate the annual growth. Thousands of years later in 1494 double-entry bookkeeping was introduced, when the Italian *Luca Pacioli* published the book "*Everything About Arithmetic, Geometry and Proportion*" [Pac94] dedicating a chapter to bookkeeping [Ale02]. It had only been two years after *Columbus* put his feet on America and two years before *Leonardo Da Vinci* started painting *The Last Supper*. Ever since, accounting has become more important and a business world without it cannot be imagined anymore. Double-Entry bookkeeping is still the standard accounting paradigm today. Accounting keeps track of the money and economic events in a company and enables detailed reporting of the finances. A couple of hundred years later the discipline of computing arose and changed our everyday life. Today, many companies rely on *information systems* (IS) which were made possible by computer science. Businesses use IS in order to capture, calculate, and analyze data as well as support the company's processes.

Bridging the areas of accounting and information systems leads to the area of *accounting information systems* (AIS) [RS11, Bod83, BSN09]. Accounting information systems capture data for resources, agents, and events involved in a company's actions. This data is analyzed, prepared, calculated, and visualized in order to provide insight into a company's current financial status as well as of its historic activities. Furthermore, it is an important tool to predict the financial future of a company and for the managers' decision making. According to [RS11] accounting information systems consist of (i) the people using the system, (ii) the processes acquiring, managing, and storing data, (iii) the data itself in regards to the business and its activities, (iv) the software enabling the data processing, (v) the systems and devices running the software and storing the data, and (vi) the control and security issues. A company's AIS was either custom made by its own information technology (IT) department or developed by an external IT company. Throughout the years customizable off-the-shelf accounting information systems by small software companies and big enterprises such as SAP became more popular and were extended to or integrated into *enterprise resource planning* (ERP) systems.

In order to create a useful and meaningful AIS, the data structure and user interface have to reflect the economic phenomena on which companies base their business. Thus, it is essential that the business people providing the requirements can unambiguously communicate with the IT professionals which are in charge of creating the system. *Business models* can be used as a language for communicating these requirements to achieve this goal. Timmers [Tim98] defines business models as "an architecture for the product, services and information flows, including a description of the various business actors and their roles". However, today there is still a lack of a business modeling language which (i) is powerful enough to capture all business activities and additional requirements to generate the IT artifacts for an AIS as well as at the same time provides (ii) an easy to comprehend representation of the business model which can be understood by business and IT professionals, respectively. Thus, the use of business modeling languages in the requirements elicitation phase of information systems as well as for the semi-automatic generation of IT artifacts for such systems is still limited. We argue, that a business modeling language which incorporates features overcoming these limitations can accelerate, streamline, and reduce costs of the AIS development process.

In this thesis, we provide such a business modeling language by creating a dedicated graphical representation for a business modeling language based on well-established concepts of economic literature called the Resource-Event-Agent (REA) ontology. We extend the language with additional concepts from the information systems discipline and use it to semi-automatically develop an accounting information systems. Additionally, we developed a modeling-tool for our new language.

1.1 Motivation

Developing accounting information systems requires business people of a company to talk to the IT people in order to collect the requirements for the system (cf. Figure 1.1). For this task, an unambiguous and easy to comprehend graphical business modeling language would be preferable. It would allow business people to precisely declare their requirements. Furthermore, a mapping between this business modeling language and a database structure would assist an IT professional in creating the IT system according to these requirements to a great extent. However, such a business modeling language combining all the aforementioned characteristics and features does not yet exist. Consequently, choosing an existing business modeling language and extending it to fit the previously mentioned requirements would fill this gap.

Nowadays, we recognize three main business model ontologies: the Business Model Ontology (BMO) [OPT05], the e³value ontology [GA01], and the Resource-Event-Agent ontology (REA) [McC82]. Among them, REA covers the most concepts based on economic literature and was built with information systems in mind. Thus, REA models are closely related to relational database structures which is beneficial when creating an AIS. Besides describing current events, REA also allows modeling commitments and agreements as well as business policies. These additional concepts are especially important for an AIS to predict future events and apply certain constraints on events. Thus, we picked it as the best candidate for capturing the requirements for an accounting information system. In general, REA provides an application-independent description of a company's economic phenomena. It has its roots in the accounting discipline and

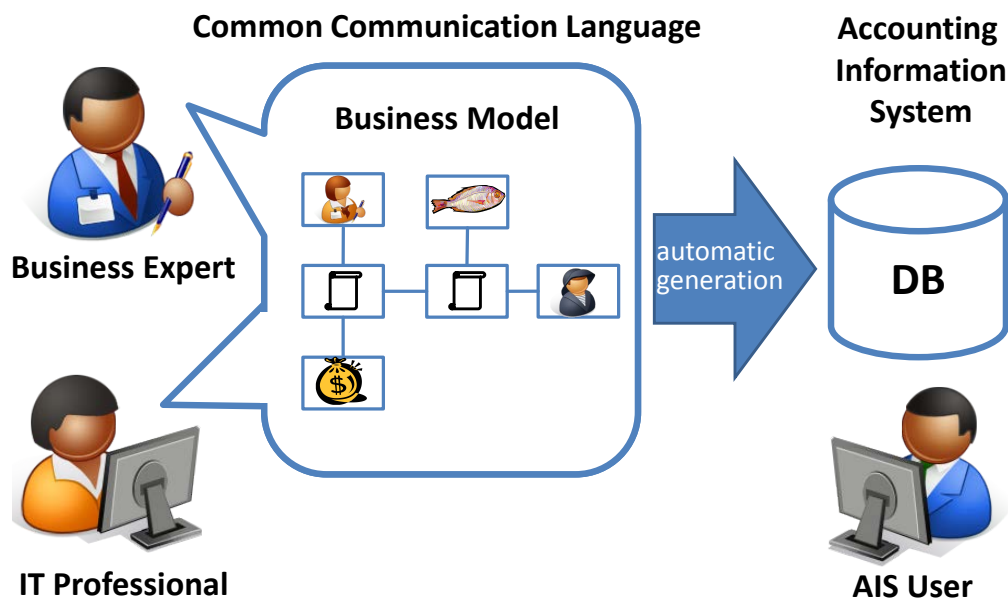


Figure 1.1: Accounting Information System (AIS) Design

is seen as an alternative accounting framework compared to double-entry bookkeeping, which is still the standard accounting framework today incorporating elements such as debits, credits, and accounts. However, whereas double-entry bookkeeping mainly shows historic data, the REA ontology can capture planned and scheduled business events as well as currently occurring events. Over the years REA was extended and evolved into a business modeling ontology [GM02] capturing a broad range of business phenomena. The representation of REA was always closely related to Entity-Relationship diagrams, thus allowing a seamless creation of database schemas for information systems.

REA spans over three different layers: (i) the operational layer, (ii) the planning layer, and (iii) the policy layer. The original REA consisted of the operational layer and defined the REA core concepts allowing to model exchanges which "have happened" or "are happening". Following its name, REA is based on the three main concepts: economic resources, economic events, and economic agents. It should be noted that for a better readability we often drop the prefix term *economic* for the remainder of this thesis. Basically, one or more resources are exchanged between usually two (but in theory also more) agents during well defined events. A cornerstone of REA is also the concept of duality, which means that usually one event (or in theory a set of events) is compensated by another event (or set of events). An example on the instance level may be: on the 30 November 2011 a sale (*event*) occurs, where the salesman Joe (*agent*) with the help of the shop assistants Mary and Wendy (*agents*) give 50 pounds of tuna fish (*resource*) and a fishing rod (*resource*) to their customer Fred (*agent*). The sale (*event*) is compensated by the payment (*event*) which happens right after Fred (*agent*) pays the amount of 700 Euros to the cashier Mark (*agent*).

Evidently, REA does not model the individual instance on M0, but analyses an enterprise on

the model layer M1. Accordingly, the REA model defines the schema for the above mentioned instance from the perspective of the seller as follows: sales and payments are in *duality*. A sale is performed by exactly one salesman, a number of shop assistants and an external customer. The sale leads to a decrease of fish and additional products. The sale is compensated by the increase in cash being part of the payment which takes place between an external customer and a cashier. In order to build this REA model, REA provides the concepts of *Resources*, *Events*, and *Agents* on the meta-model layer M2. Furthermore, the M2 layer covers the concepts of *duality* (relationship between events), *stock-flow* (relationship between event and resource), and *participate* (relationship between event and agent).

The above mentioned example demonstrates that the original REA [McC82, GM97, GM99] is capable of recording events that are happening or already have happened. As mentioned earlier, AIS should provide management an insight into a company's current financial status in order to make proper decisions. Usually, the financial status depends not only on the past, but also on commitments that the company has already made for the (near) future. Thus, it is desirable to extend REA by such commitments and internal plans to fulfill these commitments. Accordingly, REA should be able to capture the following example of a reservation: on 2 November 2011 the salesman Joe *commits* himself - as a particular salesman - and two (not yet known by name) shop assistants to give customer Fred at least 40 pounds of tuna. In return, customer Fred *commits* himself to pay a (not yet known by name) cashier. It is easy to recognize, that the above mentioned actualization of the events is in-line with the commitments, even if the actualization overfulfills the commitment by additional 10 pounds of tuna and the fishing rod. Additionally, companies also would like to describe standards and constraints on their activities through policies. One example for a policy on the instance level might be: the regular price for 1 pound of tuna in a regular sale is 10 Euro, in an international sale the price is 11 Euro. However, the actual price in a sale might vary (e.g., a discount of 10 % is given in a sale. Consequently, 1 pound of tuna only costs 9 Euro in a regular sale).

In order to accommodate such a scenario REA has been extended by a (ii) planning and (iii) policy layer [GM02, GM05, GM06]. These layers cover economic activities that should, could, or must happen in a company and are, thus, relevant for planning and controlling activities. The extension allows modeling the following facts on M1: a particular salesman makes a sell commitment that is in reciprocity to a pay commitment that is made by a particular customer. The sell commitment commits to perform a sales event in the future, where a particular known salesman and yet-not-known shop assistants give away a certain amount of fish to a particular known customer. In return, the pay commitment is an obligation that the same customer pays a certain amount of cash to the yet-not-known cashier. The policy would be modeled in the following way: a fish type has a certain price policy with a certain sale event type.

Accordingly, new concepts were introduced on the meta-model layer M2. The most central concept is a legal *commitment*. The concept *commits* associates the *agent* who does the *commitment*. *Reciprocity* is a relationship between *commitments*. The concept of *fulfill* links *commitments* and associated *events*. *Reserve* is used to associate *resources* or *agents* to *commitments*. Furthermore, REA introduces the *typification* to support the planning and policy layer.

From the above example it is evident that in some cases one may already refer to a particular, known *agent* (such as the `salesman Joe`), and in other cases one may only refer to the *type of agent* (such as `shop assistant`), because it is yet-not-known in person. This is reflected by the M2 concept of *agent type*. Corresponding concepts exist for *event types* and *resource types*. Furthermore, *policy relationships* are introduced allowing to define policies between various REA concepts (i.e., *resources*, *resource types*, *agents*, *agent types*, *event types*).

Ideally, for creating accounting information systems, business people would create business models according to REA as explained previously and IT people would take these models and transform them into database schemas serving as the basis for the accounting information systems. However, REA's class-like representation is not intuitive for business users and its vague formalization keeps space for diverging interpretations. We argue, that these limitations diminish the use of the REA ontology and therefore, REA never acquired high acceptance in the requirements elicitation phase of developing accounting information systems. We believe, that REA could greatly benefit from an easy to understand dedicated graphical representation like e^3 value [GA01] – which allows performing financial assessments of value exchanges – does. Consequently, we want to develop a domain-specific language overcoming these limitations called the *REA-DSL* and additionally provide a mapping to a relational database schema. Since we considered an approach where the data structure of the AIS is derived from a business model, the title of this thesis is `Business Model Driven Data Engineering`,

1.2 Problems and Contributions

In the above section, we have introduced the research area of this thesis as well as the motivation behind the thesis to create the REA-DSL. We have identified six problems as well as six contributions according to creating a domain-specific language for REA. The contributions are marked in Figure 1.2 by the contribution number in blue circles. In the following, we describe the problems and contributions in detail:

Problem 1: Missing formalization of the REA ontology

Since the foundations of REA were established in the original REA paper in the year of 1982 [McC82], REA has experienced many extensions in the past 30 years. Among those are the most important extensions concerning the value chain and the planning and policy layers. Three axioms (cf. Chapter 2 "State of the Art") define some restrictions for the REA ontology. However, REA always lacked a precise formalization, thus leading in diverging interpretations of the REA ontology. Furthermore, the different papers on REA use different terms to express the same things (e.g., policy layer, knowledge layer). Sometimes, the same concept is defined in different layers with different meaning. An attempt to define a common understanding has not been published so far. We argue, that a formalization for REA would help modelers to understand the REA concepts more quickly and lessen miss interpretations.

Contribution 1: REA meta-model

The first contribution of this thesis is an unambiguous meta-model for the REA ontology based on the Meta-Object Facility (MOF) [Obj11a] by the Object Management Group (OMG).

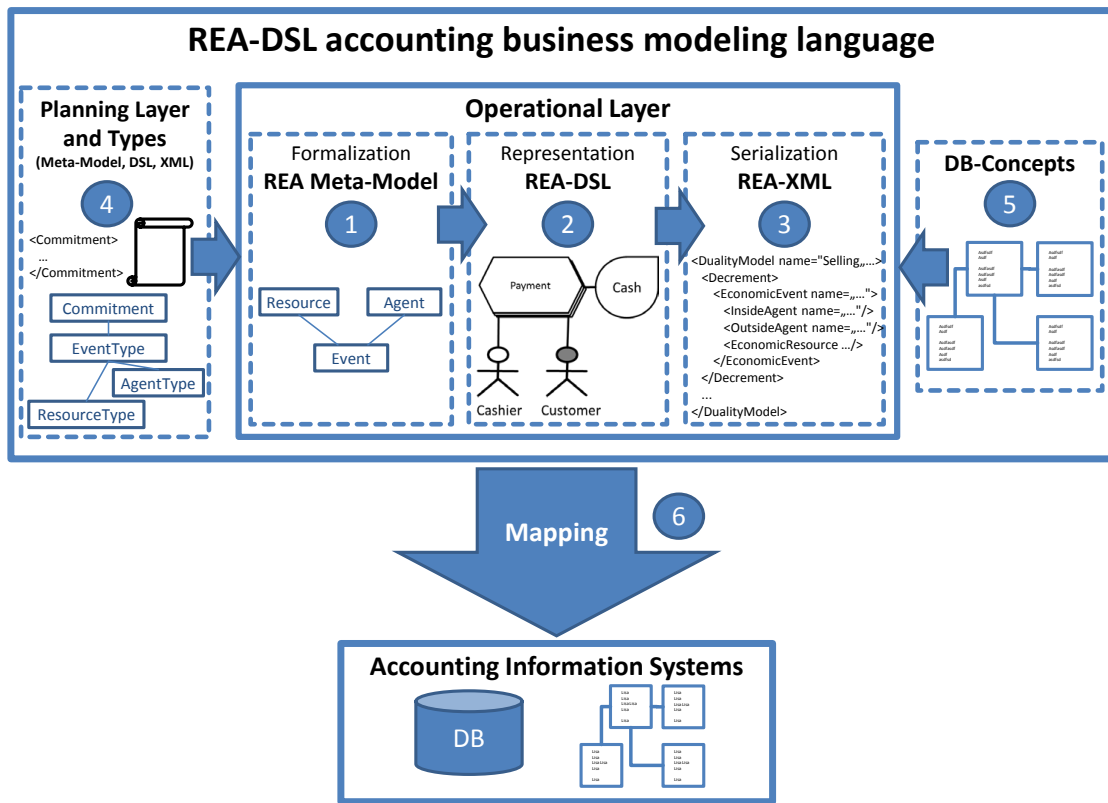


Figure 1.2: REA-DB: Contributions of this Thesis

The REA meta-model serves as a description for the actual REA models and defines constraints on them. It incorporates all concepts of the REA operational layer, which describes what economic events are actually happening. Thus, all the three core concepts – resource, event, agent – are covered by the meta-model. To reduce the complexity of the meta-model, we create a separate meta-model package for the agents (agent view), the resources (resource view), and the events (operational view). Additionally, we provide a meta-model for the value chain, which describes the economic exchanges of a REA model on an abstract level. The provided REA meta-model can be used to understand the concepts of the REA ontology and provide an unambiguous framework for the REA ontology.

Problem 2: Missing dedicated graphical notation for REA

When creating a REA conceptual data model for AISs, business professionals having the knowledge about the problem domain and IT professionals having the knowledge about the system implementation have to work together. To create models according to the REA ontology the IT and business professionals have to use a class-like or Entity-Relationship like representation. These representations are not tailored to the particular needs of the REA ontology and can grow fast in complexity with the number of REA elements defined. Additionally, these representations are not very intuitive for business professionals and hamper the communication with the IT professionals. Thus, we argue that they are a barrier for a wider acceptance and use of the REA

ontology. Different concepts of the REA ontology are presented by the same shapes and are not distinguishable at first glance. Stereotype notations provide some relief for the current REA notation. However, modeling languages such as the e³value modeling language benefit from a dedicated graphical representation having different shapes for different concepts. Consequently, we believe that a tailored graphical notation would boost the usage and acceptance of the REA ontology.

Contribution 2: REA-DSL graphical notation

In this thesis as the second contribution we provide a graphical domain-specific language (DSL) for the REA ontology called the REA-DSL. A domain-specific language is a language tailored to a specific problem area as opposed to general purpose languages such as UML or Java. The different concepts of the REA ontology are represented by own unique shapes which can be linked among each other by associations. The REA-DSL is based on the REA meta-models of Contribution 1 and thus, adheres to the constraints defined in the meta-model for the operational layer. Furthermore, we provide four different interlinked views (resource view, agent view, operational view, and value chain view) for the REA-DSL in order to split up the complexity of the REA models. Additionally, a specialized tool based on the Microsoft Visual Studio SDK helps to model the REA-DSL models and check their correctness using specified rules. Stubs of the REA-DSL can be automatically generated, which reduces recurring and time consuming tasks. An example REA-DSL model is presented using the business model of a fish retailer called Sy's Fish.

Problem 3: Missing serialization format for REA

Models of the REA-DSL cannot be exchanged between different tools because no tool-independent serialization format has been defined for the REA ontology. Thus, our REA-DSL is limited to the use of our implementation from Contribution 2 which only creates a proprietary serialization for the REA-DSL tool. Other tools, such as the implementation of the REA ontology of the University of Liechtenstein cannot import and consequently reuse our REA-DSL models. Furthermore, without a dedicated serialization format its hard to read, understand, analyze, and edit the textual representation of the REA-DSL. Additionally, the storage of the REA-DSL models in model repositories is not possible without a serialization format.

Contribution 3: REA-XML serialization language

We provide a serialization of the REA abstract syntax based on an XML schema (XSD) dedicated to the REA concepts called REA-XML. It provides a precise, tool-independent representation of REA models and may also serve as an REA model exchange language between different tools. While the structure of the serialization is defined in the XSD, the actual models are stored as human readable semi-structured XML files which are restricted by the provided XSD. Of course, we could have used an automatic transformation of our meta-model to an XML schema by the Visual Studio SDK, but this would be a hoax, since it creates a proprietary Microsoft schema including a lot of overhead not acceptable to others. Similar to the REA-DSL itself, the REA-XML is split into the four different XSDs – agent view XSD, resource view XSD, operational view XSD, and value chain XSD – to keep the XSD definitions concise.

Problem 4: Missing support of the REA planning layer and policy layer

So far, the REA-DSL (as well as the meta-model and the REA-XML serialization format) supports the core concepts of the REA ontology defined on the operational layer. This allows to create REA models capturing business exchanges which are currently occurring or which have occurred in the past. However, over the years REA was extended by the planning layer and policy layer adding concepts of policies, types, and commitments. These concepts allow to describe what economic exchanges are planned or possible. In other words, one may define contracts on future events, such as an order for a sale or the regular price for a product in a sale. Our REA-DSL is not capable of modeling these scenarios since it does not incorporate the planning and policy layer.

Contribution 4: Extending the REA-DSL by the planning view to support commitments, types, and policies

We extend the core REA-DSL by the planning layer and the policy layer of the REA ontology. Consequently, the REA-DSL is capable of capturing legally binding commitments for events being fulfilled in the future. Agents have to commit to the commitments in order to be responsible for them. Furthermore, the concepts of resource types, event types, and agent types are introduced. These concepts are especially important, if the specific resource or the specific agent is not known at the time of the commitment and will only get instantiated at the time of the fulfilling event. Another important concept introduced in this contribution is the bulk resource. Bulk resources are resources, which cannot be individually identified, or resources, where it is not economical to identify each single occurrence. Examples for such resources are nails, water, or cash. Last, we also introduce basic policy concepts for the planning view. In order to incorporate these extensions into the REA-DSL, we have to extend all artifacts created from the first three contributions. First of all, we extend the meta-model by the planning meta-model and adapt the resource, agent, operational, and value chain view. In a second step, the graphical notation of the REA-DSL is adapted and the notation for the planning view is added. In a last step, we also update the REA-XML formalization format. The tool is extended by the new concepts, and we provide an example of the Sy's Fish business model.

Problem 5: Missing database concepts for the REA-DSL

The REA-DSL is capable of capturing the most important and powerful concepts of the REA ontology. However, in order to use it for creating conceptual data models for the use in accounting information systems (AIS) it still lacks some essential concepts known from database modeling languages such as Entity-Relationship diagrams. Evidently, the REA-DSL is not capable of capturing the individual properties of agents, resources, events, commitments, stock-flows, and participate relationships. Thus, it is not possible to capture in the models which data should be recorded in the AIS. Furthermore, primary keys cannot be defined in the REA-DSL making the unique identification of runtime instances of the REA-DSL impossible. Without these concepts the REA-DSL cannot be fully used for conceptual data modeling of an AIS.

Contribution 5: Extending the REA-DSL by database concepts

We extend the REA-DSL by concepts known from the database modeling area enabling it to be used as an AIS conceptual data modeling language. Thus, we add properties to the REA concepts agents, resources, events, commitments, stock-flows and participation relationships as well as to the REA types of agents, resources, and events. Additional color coding indicates, if the property belongs to the type (yellow), to the commitment (purple), or to the actual object (green). A property can also be marked as a primary key. This indicates, that this property can be used as the unique identifier for either the agent, resource, event, or commitment. Similar to contribution five, this contribution requires the adaptation of the first three contributions. First, we extend the meta-model by properties and primary keys. Second, we define compartments attached to the shapes of the REA-DSL which can hold the multiple properties of each concept. Properties can in general be marked as primary key. Third, the properties and primary keys also become part of the REA-XML serialization language. Last, the tool and our example get extended by these additions as well.

Problem 6: Missing mapping between REA-DSL and relational model for databases

We have a fully fledged REA-DSL being powerful enough to be used by business professionals together with IT professionals for designing AIS conceptual database models. However, the REA-DSL models are just "pretty pictures" which cannot be understood or processed by database modeling tools in order to create relational models which can be used and processed by the database systems of an AIS. Consequently, the REA-DSL models still would have to be manually converted to a relational model by the IT professional. This task is very error prone and time consuming as well as includes many repeating steps.

Contribution 6: REA-DSL to relational mapping

We present a mapping between our REA-DSL and a relational model for an AIS. This mapping allows automatically generating relational models from the REA-DSL models. Therefore we take the concepts from the REA-DSL and map them to concepts from the relational models such as tables and columns. We provide a description of the mappings for the four REA views – resource view, agent view, operational view, and planning view – to the relational models by the means of rules described by pseudo code. For easier understanding, we additionally explain the mapping on the basis of a concrete example from Sy's Fish. There is no need for a mapping of the value chain view since it just provides an abstract view on the operational and planning view. The mapping is included into our REA-DSL tool by providing an automatic relational model generator. Thus, the REA-DSL tool can be used for the design of an AIS by business and IT professionals to create REA conform models on the conceptual level all the way to automatically generate a relational model IT artifact which can be processed by the AIS database system.

According to the six contributions mentioned above we define the following three hypotheses:

1. **Hypothesis 1.** *A domain-specific language of the REA ontology is easier to understand than the class-like representation.*
2. **Hypothesis 2.** *A domain-specific language of the REA ontology allows an unambiguous creation of REA ontology business models.*
3. **Hypothesis 3.** *A domain-specific language of the REA ontology can be mapped to a database scheme for an accounting information system.*

Consequently, we want to answer the following main research question in this thesis:

”How can we provide a precise and easy to understand modeling language for the REA business modeling ontology which may be used for the semi-automatic development of accounting information systems?”

1.3 Methodological Approach

In this thesis we apply a *conceptual* and *constructive* methodological approach. The general objective of such an approach is to develop models, e.g., software systems, specifications, algorithms, which go beyond the state of the art. Implementations of the systems are mostly on a prototype level. Case studies and functional tests are often used for the proof of correctness. Specifically, we will follow one paradigm built upon this approach called *Design Science*.

Hevner et al. [HMPR04] have established the design science paradigm in the *information systems* (IS) research community. The IS area is affected by people, organizations, and technology [Lee99]. Therefore, design science is not only about designing artifacts, but also about evaluating the created IT artifacts in this confluence environment. The development of REA itself actually followed a design science approach [SGM02]. Design science refers to *design processes* (production and evaluation of artifacts) and *design artifacts* (constructs, models, methods, instantiations). Usually, the production and evaluation of artifacts is an ongoing loop iterating a couple of times to improve the artifacts. There are seven guidelines for design science research which can be followed but should not be necessarily seen as a check-list [HMPR04]. In the following we provide the definition of each of the guidelines and describe how we adhere to them in our thesis.

1. **Design as an Artifact:** *Design science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation.*

In our thesis the created artifact is a domain-specific language for the REA ontology called the REA-DSL. This artifact provides a new representation and modeling language for the REA ontology. Furthermore, the main artifact is supported by additional sub artifacts, namely the REA meta-model providing a formalization, the REA-XML providing a serialization, the REA-DSL tool providing a design environment for the REA-DSL, and a mapping to a relational database schema providing the basis for an information system.

2. Problem Relevance: *The objective of design science research is to develop technology-based solutions to important and relevant business problems.*

The REA ontology itself is a mighty business modeling language based on concepts in economic literature. However, it uses a cumbersome class diagram representation and was not yet formalized leading to different interpretations of its semantics. The different interpretations are due to extensions by a worldwide community. The actual economic definitions used are very precise, especially at the operational layer, in the refereed literature. Given the missing formalization and the current representation, it is not that wide spread and did not gain much attention in the business informatics and computer science field. Consequently, it is hardly used for capturing requirements for the creation of accounting information systems. We argue, that an easy to use and unambiguous domain-specific language for the REA ontology fills this gap and leads to a higher acceptance of the REA ontology.

3. Design Evaluation: *The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.*

The contributions provided by this thesis are evaluated in four different ways: (i) the technical feasibility of the REA-DSL as well as of the relational model mapping is demonstrated by a tool implementation, (ii) a functional test based on existing REA models is conducted, (iii) expert interviews are taken, most notable with the founder of REA, William McCarthy, and (iv) a usability study is made with students. To prove the (i) technical feasibility we created a REA-DSL tool for creating REA-DSL models. Functional tests (ii) demonstrate that we can remodel existing REA models to the new REA-DSL representation and its different views without losing any semantics. Various (iii) expert interview cycles with different experts delivered us a lot of knowledge about the interpretation of REA concepts. Last, the (iv) usability study with students, REA experts, and conceptual modeling experts evaluates on the comprehensiveness of the new REA-DSL compared to the old class-like REA representation.

4. Research Contributions: *Effective design science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies.*

We create an unambiguous and discrete domain-specific modeling language for the REA ontology called the REA-DSL. Thus, we facilitate an easy to use business modeling language by splitting up the complexity of the REA ontology into different interlinked views and by creating different stencils for the REA concepts. By specifying a REA meta-model we supply a formalization for the REA ontology. The REA-XML serves as an exchange and storage format for the REA-DSL. Last but not least, a mapping to the relational database schema enables integrating the semantic of the business model into the information system's database. These contributions provide an easy to understand solution for creating REA compliant conceptual models and derived relational models for an AIS.

5. Research Rigor: *Design science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.*

In this thesis we follow these guidelines for the overall approach. Additionally, for the creation of the domain-specific language REA-DSL we follow the methodological steps that have been suggested by Strembeck and Zdun [SZ09] for the systematic development of domain-specific languages. Amongst other variants, they describe the development process for extracting a DSL from an existing system, which is appropriate for our needs, because we extract the DSL from the existing REA ontology. Accordingly, we start with (1) the identification of elements in the REA ontology. Next, we undergo a number of revision cycles of (2) deriving the abstract syntax of the REA model including the core language model and the language model constraints and (3) defining the DSL behavior, i.e., determining how the language elements of the DSL interact to produce the intended behavior. Once we have reached a stable state, we define the (4) DSL concrete syntax. Finally, we (5) implement a modeling tool support for the DSL and (6) create a mapping to a database schema to be used in an information system. Concerning the state of the art, we elaborated on work in the areas of business modeling – specifically REA –, value chain modeling, domain-specific languages and conceptual database modeling. For the evaluation of our approach we followed the evaluation methods described previously in guideline number 3.

6. Design as a Search Process: *The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.*

During the development of the REA-DSL we conducted many feedback cycles with the founder of the REA ontology – William McCarthy – as well as scholars familiar with the domain and improved the REA-DSL frequently. Concepts of the REA ontology are added to the REA-DSL iteratively to concentrate on differentiated concepts one at a time and enhance them in several improvement cycles.

7. Communication of Research: *Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.*

This research was discussed with experts in business informatics as well as experts in business modeling. The REA-DSL was intensively discussed with the founder of REA, William McCarthy. Furthermore, the REA-DSL was presented in a class of business students. Various research papers are published at renown conferences in the community and a journal contribution is intended. The publications are listed in the following:

- The REA-DSL: A Domain Specific Modeling Language for Business Models [SHH⁺11]
- A Domain Specific Modeling Language for REA [MSHH11]
- REA-XML: An Unambiguous Language for REA Business Models [MHHS11]
- Extending the REA-DSL by the Planning Layer of the REA Ontology [MH12b]
- Business-Model-Driven Data Engineering Using the REA-DSL [MH12a]

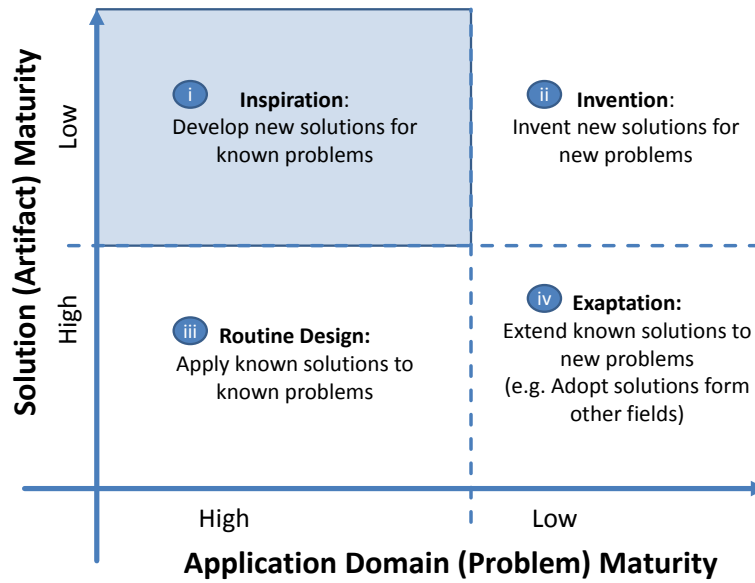


Figure 1.3: Knowledge Contribution Framework [Hev09]

Hevner also argues, that the design science research contribution can be characterized by the knowledge contribution framework [Hev09] (cf. Figure 1.3). This framework consists of two dimensions: (a) maturity of application domain and (b) maturity of solution. Depending on the relevance of these dimensions, there are four quadrants showing the significance of the research: (i) *Inspiration* - develop new solutions for known problems, (ii) *Invention* - invent new solutions for new problems, (iii) *Routine Design* - apply known solutions to known problems, and (iv) *Exaptation* - extend known solutions to new problems.

We see our research contribution for developing the REA domain-specific language in the *Inspiration* quadrant (cf. Figure 1.3 highlighted background) . First, deriving accounting information systems artifacts from business models is a known problem and second, to the best of our knowledge, there is no solution using a REA domain-specific language for designing AIS systems and deriving database structures from the REA models. Thus, we will provide such a solution by our REA-DSL and the mapping to the relational model.

1.4 Structure of this Thesis

The remainder of this thesis is structured as follows:

Chapter 2 provides an insight into the state of the art in the area of this thesis. First, we elaborate on business models in general and pick the three most recognized business models – e³value, BMO, and REA – to discuss them in more detail. Since this work builds upon the REA ontology, we will provide a thorough analyses on REA in a dedicated Chapter 3. Related work on value chain modeling, domain-specific languages and conceptual database modeling

complete this chapter.

Chapter 3 includes a thorough overview of literature on the REA ontology itself and approaches for providing a representation and formalization for REA. To the best of our knowledge, there has been no attempt on creating a domain-specific language for REA, and consequently, we are not able to provide a comparison to our approach with existing approaches. Furthermore, papers on the use of REA in information systems as well as on extending and applying REA are discussed.

Chapter 4 explains the original REA ontology and its class like representation by an example. The example is based on a real fish selling company called Sy's Fish, which was already used in previous work by Geerts and McCarthy [GM97]. We extended the example to incorporate all the concepts of the proposed REA-DSL. This example spans from buying fish from the fish market over cleaning the fish all the way to taking orders and selling the fish. We elaborate on the current limitations of the REA ontology and explain why for the new representation we chose a domain-specific language instead of a UML profile.

Chapter 5 builds the core of this dissertation by providing contribution one – the REA meta-model – contribution two – the REA-DSL graphical notation – and contribution four – the extension of the REA-DSL by the planning and policy layer. Thus, we explain the formalization and the graphical representation of the REA-DSL by using abstract example models of the five views it consists of: the agent view, the resource view, the value chain view, the operational view, and the planning view. This delivers an intuitive, unambiguous REA modeling language. At the end of this chapter, we present a complete example according to the business case introduced in Chapter 4. The content of this chapter is based on our papers: [SHH⁺11, MSHH11, MH12b].

Chapter 6 extends the REA-DSL by database concepts (contribution five) and provides a mapping between the REA-DSL and a relation model for an accounting information system (contribution six). Consequently, we adapt the meta-model by concepts of properties and primary keys. In a next step, compartments are introduced to the graphical representation of the REA-DSL to depict the properties of the REA concepts agent, resource, event, and commitment. The mapping of the REA-DSL to the relational model is described by rules using pseudo code. Additionally, the mapping is explained using a concrete example of the REA-DSL. The content of this chapter is based on our paper: [MH12a].

Chapter 7 presents the serialization format of the REA-DSL called the REA-XML as stated by contribution three. Similar to the REA-DSL in Chapter 5, the REA-XML is split up into five views described by an XML schema (XSD) for the agent view, the resource view, the value chain view, the operational view, and the planning view. The relations between the REA-DSL meta-model and the REA-XML XSD descriptions are explained. In parallel, models of the REA-DSL and corresponding REA-XML instances according to the Sy's Fish example are shown. The content of this chapter is based on our paper: [MHHS11].

Chapter 8 presents the evaluation of this thesis built upon four pillars: the technical feasibility, a functional test, experts interviews, and a usability study. We introduce the tool implemented for creating REA-DSL models and for automatically generating relational database schemas. It incorporates all the concepts presented in this thesis.

Chapter 9 concludes this thesis by summarizing the contributions and provides an outlook for future work and open issues.

State of the Art

In this chapter we discuss the *state of the art* and the *related work* in regards to Business Model Driven Data Engineering. There are four main blocks related to our REA-DSL (cf. Figure 2.1): (i) business models (including among others the REA foundations), (ii) value chain modeling, (iii) domain-specific languages (DSLs), and (iv) conceptual database modeling.

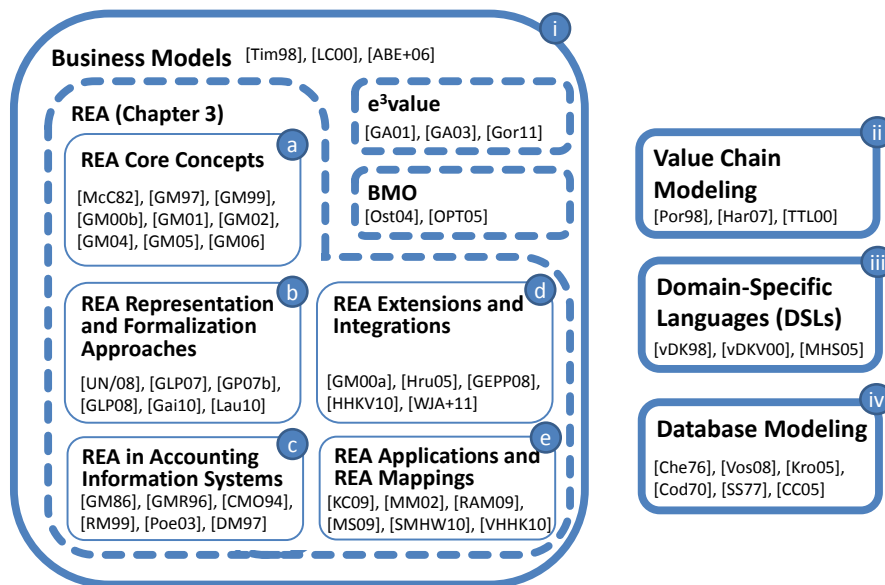


Figure 2.1: State of the Art - Overview

We elaborate on the three main (i) business models *Resource-Event-Agent ontology* (REA), *e³value*, and *Business Model Ontology* (BMO). After discussing the term *business model* we give a brief introduction of BMO and *e³value*. Since we build upon REA in our thesis, we

have a thorough study on literature in REA and therefore, provide related work in five areas of REA in a separate dedicated Chapter 3 "REA Overview". After elaborating on various business models we introduce the concepts of (ii) value chain modeling. Part of our REA-DSL is a value chain consisting of business processes involved in the company's business activities providing an overview of all the detailed REA exchanges. Furthermore, (iii) domain-specific languages (DSLs) and their benefits are discussed and potential tools are introduced. Last, we dwell on the basic concepts of (iv) conceptual database modeling in order to derive a database scheme from the REA-DSL.

2.1 Business Models

Analyzing the economic phenomena on which companies base their business may serve as a good starting point in the requirements elicitation phase when developing accounting information systems. These economic phenomena lead to business models which are defined by Timmers [Tim98] as an architecture for the product, services and information flows, including a description of the various business actors and their roles. Furthermore, it describes the potential benefits for the various business actors and the source of revenues. Linder and Cantrell [LC00] share a similar point of view and define a business model as a company's core logic in order to create value by explaining how a company acts on the market and earns money. Additionally, they specify that a business model consists of distinct components which include and represent the essential business logic building blocks. These building blocks range from revenue models and value propositions to organizational structures and arrangements for trading relationships. Andersson [ABE⁺06] argues, that business models usually specify – amongst other things - the main actors, their relationships and the values exchanged between them. According to Amit and Zott [AZ01] a business model depicts the content, structure, and governance of transactions designed so as to create value through the exploitation of business opportunities. Transaction content is seen as the exchanged goods or the exchanged information. Transaction structure refers to the involved parties participating in the exchange, their linkage among each other, and the sequence of the exchanges. Transaction governance is responsible for the control of flow of information, resources, and goods by the relevant parties.

We see three main ontologies to conceptualize business models: the Business Model Ontology (BMO) [OPT05], the e³value ontology [GA01], and the Resource-Event-Agent ontology (REA) [McC82]. BMO is easy to use by the domain expert because it focuses mainly on the categorization of aspects relevant for the delivery of products and services to fulfill customers' requests. It helps the domain expert to ask herself the right questions when developing a business model, but has a limited focus on conceptualizing the elements of the business model. In contrary, e³value defines a conceptual model to describe the exchanges of values among actors in a network [GOP05]. e³value comes with a graphical syntax that is easy understood by the domain experts. Furthermore, it allows the domain expert to perform financial assessment of the value exchanges. However, it is not capable of describing commitments and policies and thus, it is not able to define planned and standard economic exchanges. REA has its roots in the accounting discipline and is based on strong concepts of the literature in economic theory [Yu76, Por98]. Additionally, REA focuses on IT implementation issues and is closely related

to Entity-Relationship (ER) models, which makes it a good choice to use in a business model driven data engineering approach. An overview of the introduced business models is given in [Ila07]. Additionally, a business modeling reference ontology inheriting the concepts of these three business ontologies is presented in the aforementioned thesis. In the following two subsections we elaborate on BMO and e³value. Since we do a thorough and deep analysis on REA, a separate chapter is dedicated to the REA related work in Chapter 3 "REA Overview". Thus, the last section just contains a reference to this chapter.

2.1.1 BMO - Business Model Ontology

The Business Model Ontology (BMO) builds upon existing knowledge in business modeling and was introduced by Alexander Osterwalder in his thesis in 2004 [Ost04]. As distinguished from e³value and REA, it focuses on the viewpoint of one company within a business network and the company's profitability. It represents the value given to customers by modeling business concepts and their relationships. BMO contains four main perspectives called pillars partly based on the Balanced Scorecard approach by Kaplan and Norton [KN05]. These pillars are:

- **Product:** What products are offered to the market?
- **Customer Interface:** Who are the customers?
- **Infrastructure Management:** How to perform logistical issues?
- **Financial Aspects:** What is the revenue model?

Furthermore, Osterwalder splits these four perspectives into nine interrelated building blocks representing the core of the ontology. These nine building blocks, which are based on various existing business modeling literature [LC00, WV01, AT01], are:

- **Value Proposition:** Products and services that are of value to the customer.
- **Target Customer:** A segment of customers a company wants to offer value to.
- **Distribution Channel:** How to reach the customers?
- **Relationship:** Kind of link between customer and company.
- **Value Configuration:** Alignment of activities and resources to create value for the customer.
- **Capability:** Ability to execute a repeatable pattern of actions.
- **Partnership:** Cooperative agreement between companies.
- **Cost Structure:** Representation in money of all the means employed in the business model.
- **Revenue Model:** How the company makes money?

The Business Model Canvas

Sy's Fish

Designed for: _____
 Iteration: _____

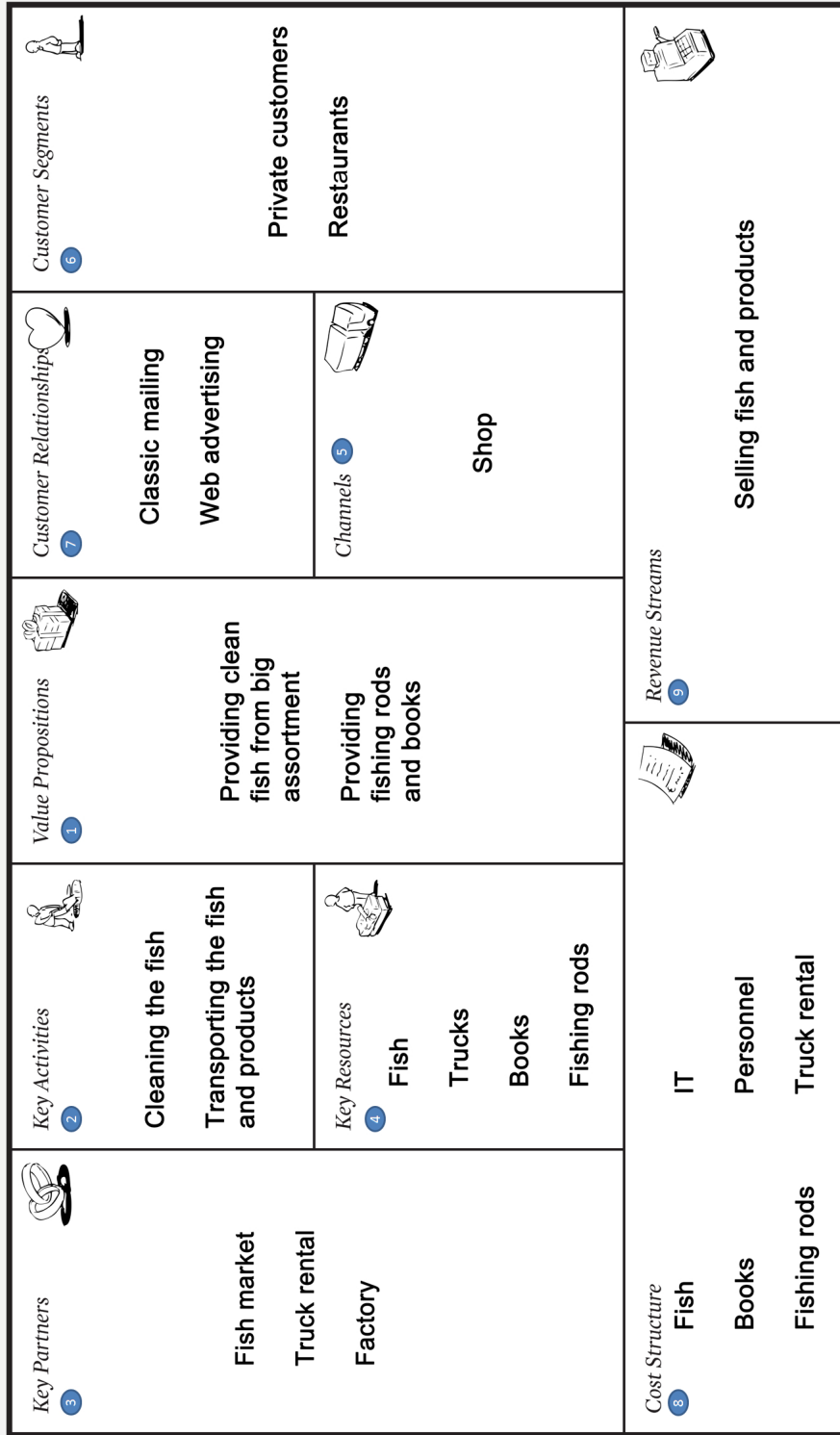


Figure 2.2: DSL Approach

In Figure 2.2 a BMO model for the fish selling company Sy's Fish is shown on the Business Model Canvas. The business model canvas allows companies to describe their business model on a big canvas showing the nine building blocks. Notice, descriptions of the building blocks in the canvas have changed slightly from the original work by Osterwalder [Ost04]. The value proposition (1) of Sy's Fish is to provide the product cleaned fish from a big assortment, as well as providing fishing rods and books. These fish and products are of special value for the customers. The infrastructure management, consisting of the key activities (2), key partners (3), and key resources (4), describes how Sy's Fish is able to acquire and clean the fish, in order to be able to sell the cleaned fish. Therefore, Sy's Fish has the key activities/capabilities (2) to clean fish and to transport fish and products. He has key partners/partnerships (3) with fish markets where he can buy the fish, the truck rental station, and the factory for purchasing the products. In the business modeling canvas, the key resources are depicted instead of the actual value configuration. The key resources (4) are fish, trucks, books and fishing rods. The customer interface explains, how Sy's Fish is able to advertise his products and how to sell them to his customers and consists of the channels (5), the customer segments (6), and the customer relationships (7). Channels (5) to sell the clean fish are the shops where private or restaurant customers (6) buy fish. A relationship (7) to the customers is kept using classic mail and web advertising. Concerning the financial aspects, the cost structure (8) consist of fish, books, fishing rods, IT, personnel, and rent payments for the trucks. At the end, Sy's Fish makes a revenue (9) by selling the fish and products which will lead to either a profit or loss.

As you can see, the BMO model in Figure 2.2 shows Sy's Fish strategy to make a profit. We argue, that among the discussed business models, BMO has the strongest focus on the company's strategy. However, it does not provide a detailed description of the data structure including its properties.

2.1.2 e³value

e³value [GA01] describes from an observers perspective a network of multiple actors creating, exchanging, and consuming values in order to satisfy a consumers need. Its primitive concepts are not novel. They are based on the exact same REA ideas published more than 20 years before. Only the graphics are new. e³value is based on economic reciprocity, where a business partner providing values to another partner receives objects of value from this partner. For example, Sy's Fish is selling fish (giving a value) in order to get money in return (getting a value). Values are the main artifacts and can for example be products, services, rights, or intangible assets (e.g., better image of the company or the good feeling someone gets after donating to a good cause). e³value depicts the economic exchanges between multiple actors on a rather high level of abstraction answering the question who is doing business with whom exchanging what. One of the strengths of this business ontology is certainly the dedicated graphical representation which consists of a small sets of elements and their relations [GA03] as well as proper tool support [Gor11]. Additionally, the profitability of a network can be calculated and expressed in profitability sheets. However, e³value does not expose the internal processes of a company and hence, does not concentrate on implementation and alignment issues of an IT system.

The e³value model in Figure 2.3 shows the value network of a fish selling company called *Sy's Fish*. The main business idea is to buy fish from a fish retailer and sell it to customers. The

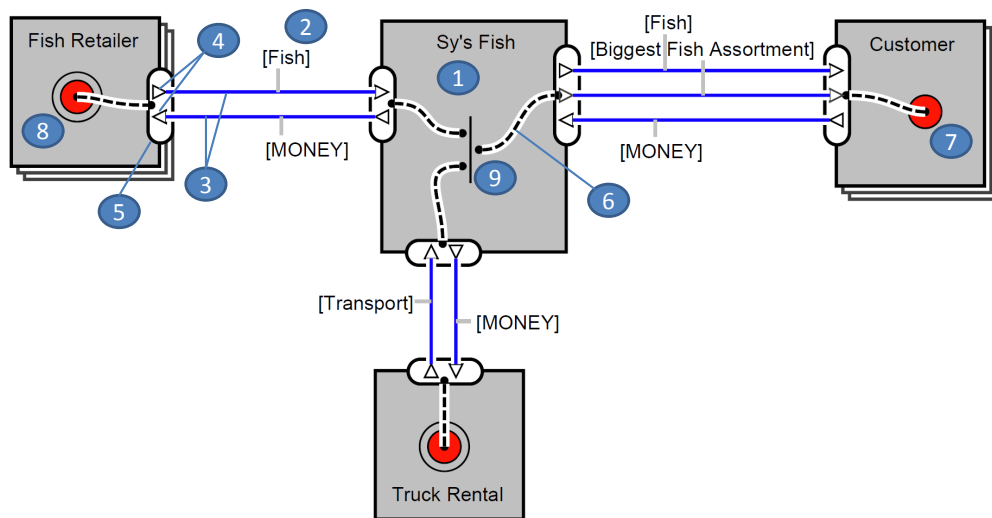


Figure 2.3: e³value Model

fish is transported by rented trucks. An e³value model consists of various modeling elements (noted by numbers in circles): *Actors* (1) (such as Sy's Fish) are represented by rectangles. A stack of rectangles (e.g., customer) is a specific actor referring to a market segment. *Value objects* (2) are exchanged between the actors/market segments in order to make profit. They can either be goods (e.g., fish, money), services (e.g., transportation), or intangible goods (e.g., biggest fish assortment). A *value transfer* (3) delivers the value object from one actor to another and is denoted by a line. *Value ports* (4) specify, if a value object is received or provided, depending on the direction of the triangle (e.g., fish transferred from fish retailer to Sy's Fish). The *value interface* (5) bundles multiple value ports. All value transfers starting/ending in the same value interface have to occur together (e.g., Sy's Fish only gets fish if he pays money). *Scenario paths* (6) are dashed lines and show the flow of the value objects between the value interfaces inside an actor. The scenario path starts at the *start stimulus* (7) and ends at the *stop stimulus* (8). *AND* (9) forks (depicted by a bar) and *OR* forks (depicted by a triangle - not shown in this figure) can split the scenario path (e.g., in order to sell the fish to the customer, we need to get the fish from the fish retailer and also transport it with the truck from the truck rental).

2.1.3 REA

As mentioned earlier, REA is the business ontology of choice for our business model driven data engineering approach. Consequently, we dedicate a separate chapter (cf. Chapter 3 "REA Overview") to it and will not go into more detail here.

2.2 Value Chain Modeling

Part of our REA-DSL is the modeling of value chains which is an abstract view of the core REA concepts of the process specification layer. Therefore, we elaborate on the definition of value chains in this section.

A value chain captures the activities through which companies can create competitive advantage. A value chain is a sequence of activities adding value to a product passing through them. At the end, the value chain is expected to create value for the customer which is higher than the combined cost of all activities, thus providing a profit margin to the company [Por98]. REA also makes use of value chains where the value chain processes are connected by resource flows. These value chain processes reside on the value chain specification layer and can be decomposed into matched core REA patterns on the process specification layer [GM97]. Originally, in 1985 Porter introduced in his book *Competitive Advantage* [Por98] a generic value chain model consisting of a set of activities found in a wide range of companies. Each of these activities is supposed to add value to the product and may be vital to competitive advantages. He identifies five primary activities:

1. **Inbound Logistics.** Involves the receiving, storing, inventory control or return of input materials.
2. **Operations.** Involves transforming input materials into the final product. This includes production, packaging, equipment maintenance, testing, and facility operations.
3. **Outbound Logistics.** Involves storing the finished products, order processing, and scheduling as well as distributing the final product to the customers.
4. **Marketing and Sales.** Involves pricing and advertising the product in order to induce the customers to purchase the product.
5. **Service.** Involves the support of the customer after the product is sold (e.g., installation, repair, training)

Each of these five activities can include specific activities which vary by industry. Additionally, four support activities facilitate the primary activities: firm infrastructure, human resource management, technology development, and procurement. All these activities also affect the profit margin and therefore, the competitive advantage of a company. This advantage can be a cost advantage (i.e., being cheaper than the competitors) or a differentiation (i.e., being unique at something the customer values).

Harmon [Har07] defines a value chain as follows: "A very large-scale business process that is initiated by a customer request, or by the decision of the company to enter a new line of business, and results in the delivery of a process or service to a customer. A value chain includes everything that contributes to the output. By adding up all of the costs of each activity in a value chain, and subtracting the total from the sale price, an organization can determine the profit margin on the value chain."

In his handbook for value chain research, Kaplinsky [Kap00] defines a value chain as "describing the full range of activities which are required to bring a product or service from conception, through the different phases of production (involving a combination of physical transformation and the input of various producer services), delivery to final consumers, and final disposal after use". Interestingly, he does not use the notion of generating profit in his definition. A big focus in this handbook is put on value chains concerning globalisation.

E-commerce had a big burst in the last decade and, therefore, changed the internal and external conditions of enterprises. In [GY08] the shortcomings of traditional value chains are discussed and an extended e-commerce value chain based on Porter's value chain is proposed. These changes are supposed to increase the competitive advantages of e-commerce related companies.

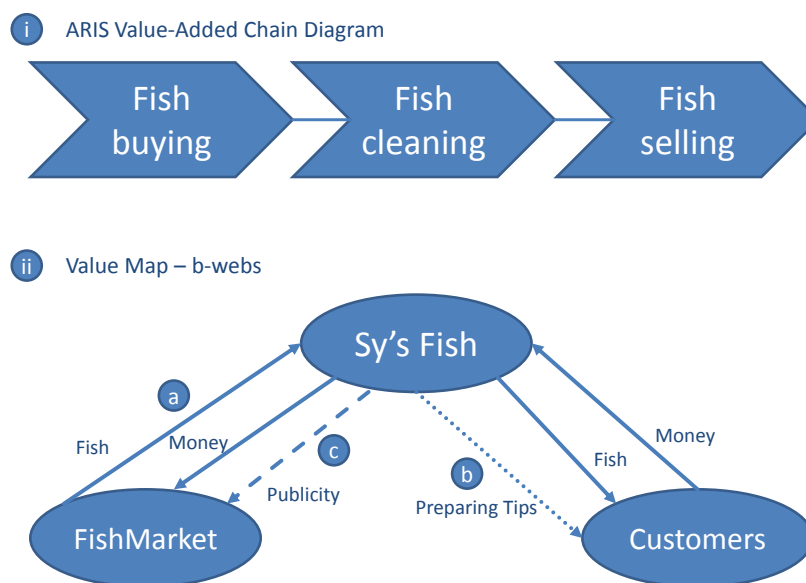


Figure 2.4: Value Chain Notations

Various notations exist for value chain modeling. The *ARIS* platform [Sof11], which has been developed by the German company *IDS Scheer* – later acquired by *Software AG* – provides the *value-added chain diagram* (VAD) [DB07] (cf. Figure 2.4 (i)). It consists of *function* objects (denoted by an arrow like shape) which are connected to a function sequence. An example of such a function sequence is the buying, cleaning, and selling of fish. Tapscott [TTL00] uses *value maps* (cf. Figure 2.4 (ii)) to depict his business webs (b-webs). Ellipses represent the involved partners. There are three different kind of flows: (a) goods, services, revenue depicted by a solid arrow, (b) knowledge depicted by a dotted arrow, and (c) intangible benefits depicted by a dashed arrow. *Sy's Fish* sells the good fish to the customer. Additionally, he provides some tips for preparing the fish. These tips are intangible benefits for the customer. In return, *Sy's Fish* gets money from the customer. Another modeling language is currently in development by the OMG. It is named *value delivery modeling*

language (VDML) [Obj11d] and will provide value chain modeling capabilities. Ideas from e³value and REA will be incorporated into the language. The value chain notation of Geerts and McCarthy from 1997 [GM97] was based nominally on Porter in [Por98] originally published in 1985. Geerts and McCarthy altered that notation to include specific resource flows between processes plus the decrement–increment notation. They also eliminated the idea of support activities, preferring instead to start with the notion of a completely specified value chain. This is something Porter did not aspire to.

2.3 Domain-Specific Languages

In order to create a dedicated language for REA we need to use a tool allowing us to describe the language. Different approaches to create such a domain-specific language and to generate an editor for using the generated languages is discussed in this section. First, we introduce the meaning of domain-specific languages.

A Domain-Specific Language (DSL) offers expressive power through appropriate notations and abstractions focused on – and usually restricted to – a particular problem domain [vDKV00]. We see an increasing interest in domain-specific modeling languages [KT08, GSCK08] based on dedicated meta-models and notations. van Deursen et al. [vDKV00] claim the following benefits of a DSL approach: They allow solutions to be expressed at the level of abstraction of the problem domain. As a matter of fact, domain experts themselves can understand, validate, and often modify DSL programs/models. The DSL programs/models are concise and self documenting to a large extent. They enhance productivity, reliability, and maintainability. DSLs allow for validation and optimization at the domain level. DSL models may be used to derive machine processable artifacts. These generated artifacts include program code or descriptive definitions such as database schemas for information systems. Concepts related to DSLs date back to 1976, where Parnas introduced a similar approach called program families [Par76]. A program family is a set of related programs (i.e., they are in a specific domain) which share certain commonalities and can be used to create further software in this domain. An approach to integrate program families and DSLs is proposed by Smith et al. [SMC07]. In 1982 Bentley [Ben86] had an early idea of domain-specific languages. He argues, that the work of programmers can be seen as constant inventions of little languages which are targeted at solving problems in a specific area.

Most domain-specific languages are small textual and usually declarative languages. Two types of DSLs exist: textual DSLs and graphical DSLs. A textual DSL represents domain-specific characteristics and relationships between the different characteristics in a textual manner. Compared to a general purpose language such as C or Java, which address problems of a wide area, a textual DSL focuses on a well defined problem area. Since textual DSLs are less powerful than graphical DSLs, we do not consider them in our approach. A graphical DSL, compared to a textual DSL, provides an intuitive and less susceptible to error approach to creating valid concepts of the domain. As from now on, we will in general use the term DSL to refer to a graphical DSL.

When developing a DSL, the designers usually have to undergo following steps [vDK98, vDKV00, MHS05]: (1) identify the problem domain, (2) gather all relevant knowledge in this domain, (3) cluster this knowledge in a handful of semantic notations, (4) design a DSL that

concisely describes applications in the domain, (5) construct a library that implements the semantic notions, (6) design and implement a compiler that translates DSL programs to a sequence of library calls, and (7) write DSL programs for all desired applications and compile them.

Various approaches and tools to create DSLs exist. The interested reader may be referred to the Domain-Specific Modeling (DSM) forum site [Dom12] for an overview of the different tools. We briefly elaborate on the most common approaches: (i) Unified Modeling Language (UML) Profiles (although, in fact it is not a DSL), (ii) Eclipse Modeling Framework (EMF) together with the Graphical Modeling Framework (GMF), and (iii) Microsoft Visualization and Modeling SDK.

UML Profile. First, a (i) UML Profile is actually not a DSL, but can be used to accomplish similar goals. UML [Obj11c, RJB10] itself is a general purpose language introduced by the Object Management Group (OMG) conforming to the Meta Object Facility (MOF) [Obj11a]. Opposed to DSLs, which define a completely new language, UML Profiles apply certain restrictions like stereotypes, tagged values, and OCL (Object Constraint Language) constraints [Obj12] to existing UML models in order to fit them to a specific domain. In a previous project on UN/CEFACTs (United Nations Center for Trade Facilitation and Electronic Business) *Core Components* [UN/09] we have created both, a *UML Profile* and a *DSL for Core Components* [LM09]. We have identified a set of advantages of the DSL approach compared to the UML Profiles approach. Thus, we also opted for using a DSL for REA ontology modeling. Table 2.1 shows the most important differences between UML and a DSL.

	UML	DSL
Dedicated	+/-	+
Extensible	+/-	+
Restrictable	+/-	+
Shape adaptable	-	+
Processable	+/-	+
Holistic	+/-	+

Table 2.1: UML vs. DSL

- **Dedicated.** DSLs and UML are both used for modeling, but they aim at different problem areas. UML may in principle be used to depict any problem scenario with regard to its application structure, behavior, and architecture, as well as its business processes and data structures. In contrast, a DSL leaves out unnecessary aspects and focuses entirely on a specific problem domain, in our case the business modeling domain. Thus, DSL based solutions are streamlined and less complex than their UML equivalents.
- **Extensible.** The UML is defined on the meta-model layer (M2) according to the Meta Object Facility [Obj11a]. Consequently, the concepts defined in the UML meta-model [Obj11b] are used to build UML models on the model layer (M1). In principle, anybody may adapt the UML meta-model using the concept of UML profiles in order to customize the generic UML meta-model to a specific application domain. However, UML profiles are still limited in their expressiveness, because they must adhere to the UML meta-model specification. In contrast, a DSL defines its own meta-model and must not consider any predefined limitations. As a result more powerful domain-specific solutions are feasible.

- **Restrictable.** As part of its profile mechanism, UML provides the Object Constraint Language (OCL), which allows restricting the UML meta-model in a formalized manner. However, most of the currently available UML modeling tools cannot interpret OCL. If extension mechanisms of the UML modeling tool are available, OCL interpreters or validators checking the OCL constraints against a UML model must be implemented manually. However, such an approach requires considerable coding effort. In contrast, DSL definitions allow the definition of customized code, which specifically restricts a DSL. These code fragments, checking the consistency of a DSL based model, are relatively easy to implement. Using the built-in validation feature of the DSL, a user is, for instance, prevented from nesting core components recursively.
- **Shape adaptable.** In regard to the graphical representation of a UML model, the UML modeler is limited to a well defined set of shapes representing classes and packages. Although the UML meta-model would in principle allow using different shapes for stereotypes, most UML modeling tools restrict the set of allowed shapes. Although this may be regarded as a benefit, since all modelers have a common understanding of elements by visually recognizing their purpose, domain-specific amendments are not possible. In contrast a DSL, using the built-in shape mechanism, allows any visual representation of choice for elements and associations between those elements. Thus, any domain-specific realization in regard to visual representation of concepts is possible.
- **Processable.** The overall goal of a model-driven approach is the generation of code artifacts (e.g., relational table schema) from a platform independent models (e.g., a conceptual REA ontology business model). In UML, user specific code generators built on top of UML tools, which are complex and expensive to implement, must be realized. Some UML case tools do not even allow direct access to the tool-internal representation format of the UML model. In contrast a DSL store may be easily accessed using a programming language. With the T4 template mechanism an additional powerful transformation feature is provided.
- **Holistic.** A holistic model-driven development requires a set of different features such as transformation mechanism between different model types, for example platform independent models (PIM) and platform specific models (PSM). Additionally, the transformation of model representations to code representations, such as relational database schemas, must be realized. Although all these features are in principle possible with UML, considerable coding effort and the integration of different tools for the specific tasks is necessary. In contrast, a DSL may be integrated into a Software Factory which comprises code generators, different domain-specific languages as well as wizards, guiding a modeler through transformation tasks. Thus, the modeler is provided with a holistic solution approach for a domain-specific problem.

Considering all the advantages of the DSL compared to the UML profile we have identified in [LM09], as well as the experience gained with both approaches in former projects, we decide to use a DSL instead of a UML profile for creating a dedicated graphical modeling language for REA.

EMF. Second, (ii) EMF [SBPM09] is based on the meta-modeling language Ecore which is similar to MOF. It is part of the popular Eclipse platform and can be used to specify domain-specific languages by defining a meta-model using a graphical editor. Additionally, GMF allows assigning shapes to the different model elements. The meta-model can then be used to generate a JAVA implementation of a graphical DSL editor allowing users to create specific DSL model instances. These model instances can be saved using the XMI (XML Metadata Interchange) [Obj10] standard. A Mapping between EMF and MOF is provided by [MRG07].

Microsoft DSL. Third, the (iii) Microsoft Visualization and Modeling SDK (VMSDK) – formerly known as Microsoft DSL Tools [CJKW07] – offers the development of DSLs integrated in the widespread Visual Studio development environment. It offers a single integrated view of meta-modeling the DSL and assigning diagram elements to it. The generated graphical DSL editor is based on C# and can easily be extended by overriding methods. Constraints can be applied by using rules or adding them directly to the generated code. A model store enables navigating through the DSL model instances and generating artifacts by writing custom code in any .NET language. Another approach to create artifacts is the Text Template Transformation Toolkit (T4), which allows defining text templates. Text templates may include processing directives, text blocks, and code blocks referencing the DSL model. Code blocks may be written in C# or Visual Basic .NET. Model instances can further be saved in XML conforming to an XML schema. Figure 2.5 gives an overview of the Microsoft DSL approach.

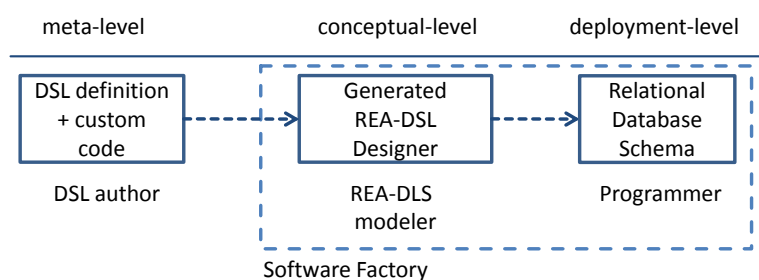


Figure 2.5: DSL Approach

First, a DSL modeler creates the DSL definition together with custom code on the meta level. In case of our REA ontology approach, the custom code includes definitions necessary for the validation of the REA model. In a second step the DSL definition from the meta level is used to generate a REA-DSL designer. The designer only contains those modeling elements, necessary for the assembly of REA-DSL models. A REA ontology modeler uses the REA-DSL designer to model REA ontology compliant REA-DSL models as shown in the center of Figure 2.5. Using the built-in template mechanism of the Microsoft VMSDK, the DSL based REA model may be transformed to relational database schemas for the deployment level. Thus, the DSL approach perfectly fits our requirements to first define a REA ontology model on a conceptual level and secondly use the conceptual model to derive relational table schema artifacts for an accounting information system.

A comparison between using VMSDK and EMF is given in [Ö07]. The author mentions as benefits of the VMSDK its stability and the tight integration with the popular development

environment Visual Studio. By using the two tools in former projects, we also discovered, that the usage of adding shapes to the DSL is a lot more intuitive and robust in VM SDK than in EMF. Given these experiences, we decided to use VM SDK for our REA-DSL implementation.

2.4 Database Modeling

The ultimate goal of the REA-DSL is to (semi-)automatically create an accounting information system based on the REA models. An information system (IS) is the integration of business related artifacts with information technology in order to support business activities and reach business goals. It stores and transfers information about processes and entities of businesses. This for example includes human resource management, enterprise resource planning, information processing, product distribution, or information analyzing. An effective information system is nowadays a key component for running an efficient company. In order to store all the relevant data, an important component of an information system is the data storage. There are various approaches for data storage including file systems, relational databases, object-relational databases, XML-databases, or NoSQL databases, which are lately gaining importance [Lea10]. However, the most widely used systems by companies today for AIS are relational databases [Vos08]. Originally, REA was created with relational databases in mind. This is due to the fact, that an AIS usually relies on relational database modeling. Consequently, we generate relational databases derived from REA for the data storage of the accounting information system.

A database (DB) is a logical integrated data pool, which is managed by a database management system (DBMS) and is described by a datamodel [PU03]. According to Kroenke, data modeling is the process of creating a logical representation of the structure of a database [Kro05]. It has to reflect the real world artifacts and their relations, which need to be stored in the database. The most common data models used for database design are the Entity-Relationship diagram (ER) and the relational model. First, a conceptual ER diagram is created describing the ontology of a specific domain. This ER diagram is then mapped to a logical relational model, which can then be used to implement the database.

Entity-Relationship Diagram. The ER diagram was first introduced by Peter Chen in 1976 [Che76]. It helps the database designer to gain knowledge about the objects and associations needed in the database by creating a semantic model. The three main elements of an ER diagram are: (i) entities, (ii) attributes, and (iii) relationships. (i) Entities (depicted by a rectangle) are the real world objects a database has to store information about. These can for example be people, facilities, events, products, or accounts. The term entity is used in ER diagrams to refer to entity types and not entity instances. Entity types refer to a group of related objects (e.g., fish are animals that live in the water), whereas an entity instance is a single occurrence of an entity type (e.g., the five pound fish I caught yesterday). (ii) Attributes (depicted by an ellipse connected to the entity) contain characteristics about the entity. Considering the entity fish, the attributes may be size, weight, color, or price. There must be at least one attribute or a combination of attributes serving as the primary key to uniquely identify an entity instance. (iii) A relationship (depicted by a diamond) associates one entity with another. For example, the two entities fish and person are connected by the relationship owns. This indicates, that a person can own a

fish. Relationships have cardinalities indicating the number of references between the entity instances. One-to-one (1:1) relationships for example mean that one person can only own one fish and one fish can only be owned by one person. One-to-many (1:n) relationships allow a person to own as many fish as he wants to, but one fish can still be owned by only one person. In a many-to-many (m:n) relationships a fish can be owned by multiple persons and one person may own multiple fish. In order to create a database, the introduced conceptual ER diagram has to be mapped (cf. [FV95]) to the logical relational model discussed next.

Relational Model. The relational model was presented by Codd in 1970 as a first-order predicate logic [Cod70]. Data is represented by mathematical n-ary relations and are kept consistent by strict rules. A relation can be seen as a table containing the data of an entity. An element (i.e., the entity instance from an ER model) of a relation is called tuple and corresponds to a row in the table. Each column of the table represents an attribute of an entity conforming to a specific data type. One cell represents the attribute value for a specific entity instance. The header of the table is the relation schema, i.e., the sum of the attribute types. For example, we transform the fish entity from the ER diagram example above into a relational model. A table representing the fish entity exists of the columns size, weight, color, and price, which correspond to the fish entity attributes. Each row in the table represents an individual fish with its specific attribute values. One row entry may be a fish with the size one feet, the weight five pounds, the color gray, and the price seven dollars.

SQL [DD97] developed by Chamberlin and Boyce [CB74] is certified by ISO [ISO11a] and may be used to make these relational models available to the relational database systems. SQL is a declarative domain-specific programming language to define, modify and query data of relational database systems. Most important for us, we can use SQL statements to define the relational tables and their columns we generate from our REA-DSL models. These SQL statements can then be used to load the relational models into a relational database modeling tool or directly feed the relational database system with the statements to create the physical tables.

Object-Relational Mapping Strategies. Aggregation and generalization in database design is discussed by Smith and Smith [SS77]. This is important, once we derive the database tables from the REA-DSL. First, we have to figure out which tables we might be able to aggregate during the mapping process. Second, since we also use generalization in REA, we need to find an appropriate representation for it using database schemas. As for the mapping strategies between Entity-Relationship models and relational models, we refer to basic database literature [PU03, Vos08]. Additionally, hierarchy mapping strategies are discussed in [BPS94, FV95, CC05, LG07].

As depicted in Figure 2.6, there are mainly three different mapping strategies between object hierarchies and relational models [CC05]: (i) single relation inheritance, (ii) class relation inheritance, and (iii) concrete class relation inheritance. In the (i) single relation inheritance the hierarchy is represented by one relation in the relational model. This relation consists of all attributes of all classes in the hierarchy. An additional type attribute identifies the original class of the hierarchy. In the (ii) class relation inheritance, every class from the hierarchy becomes its own relation with its own attributes, the more specific class contains the foreign key to the more

general class as primary key. Last, the (iii) concrete class relation inheritance creates for each concrete class an own relation with all attributes. Each relation will also include the attributes of its super class.

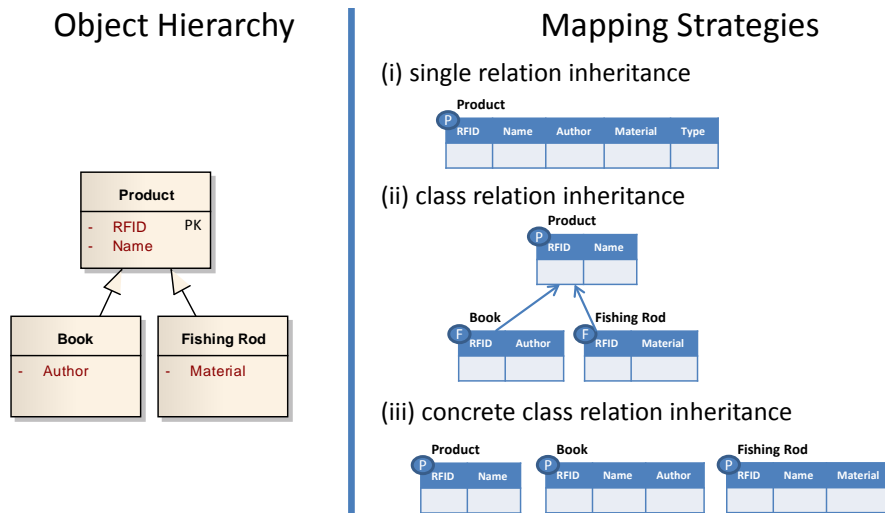


Figure 2.6: Object-Relational Mapping Strategies

REA is built with the use of relational databases in mind and therefore, its current graphical representation is similar to ER models. However, this ER representation is hard to understand by business people. Therefore, we create an easy to understand domain-specific language for REA and provide an automatic mapping to the database specific relational model, which can be used for the database creation.

REA Overview

This chapter covers the related work on REA. As already mentioned before, we decided to use REA as the business modeling ontology for our business model driven data engineering approach. Points in favor are the strong background of REA on literature in economics as well as concepts related to relational database systems. It is nowadays taught in database and accounting information systems classes in more than 100 schools and faculties [McC03]. Several books concerning the REA ontology have already been published. The more thorough ones are: *Model-Driven Design Using Business Patterns* by Hruby [Hru06], *Enterprise Information Systems: A Pattern-Based Approach* by Dunn et al. [DCH05], *Accounting, Information Technology, and Business Solutions* by Hollander et al. [HDC99], and *Modeling and Designing Accounting Systems: Using Access to Build a Database* by Chang et al. [CI06].

We provide related work on REA in five areas: (a) *REA core concepts* cover the fundamentals of the REA ontology as introduced by McCarthy and Geerts. It provides the basis for our REA-DSL. A more educational-focused description of the REA ontology and an REA example is not subject to this chapter but is provided in Chapter 4. In (b) *REA representation and formalization approaches* we discuss different proposals for representing or formalizing REA. Since we create our own formalization and representation we explain how we distinguish from these related ones. (c) *REA in accounting information systems* gives an overview of REA tool support for creating information systems and of database related issues. The ultimate goal of our REA-DSL is to create information systems. The section on (d) *REA extensions and integrations* elaborates on ideas to integrate the REA ontology with other concepts as well as proposing extensions/changes to the REA ontology. In our work we need to integrate concepts from the database area in the REA-DSL in order to derive relational tables for the accounting information systems. (e) *REA applications and REA mappings* provides an overview of work using REA as well as work on mappings between REA and other languages. Mappings between the REA-DSL and DSLs created for e³value or the UN/CEFACT's Modeling Methodology (UMM) [UN/10, ZLS08] can serve as important artifacts to be used in a top-down approach for developing inter-organizational systems [Zap09].

3.1 REA Core Concepts

The Resource-Event-Agent ontology (REA) was initially introduced by McCarthy in the fields of accounting [McC79,McC82]. Its common goal is the application-independent description of a company's economic phenomenas. In its beginnings it was targeted at modeling basic accounting information systems and was closely related to Entity-Relationship (ER) representations. REA initially focused on the concepts of economic exchanges and conversions of the present and past. Before the advent of REA, double-entry bookkeeping was and still is the standard accounting framework. It includes elements such as debits, credits, and accounts, which are artifacts associated with journals and ledgers. Thus, they are simply used for manually storing and transmitting historic data but are not essential aspects of an accounting information system capturing planned and occurring exchanges [McC82]. Therefore, REA does not build upon double-entry bookkeeping concepts (contrary double-entry bookkeeping applications may be based on REA though) and focuses on strong concepts of literature in economic theory [McC82]. REA evolved over the years and was extended to a business modeling ontology [GM02]. According to Gruber [Gru93], an ontology is "an explicit specification of conceptualization".

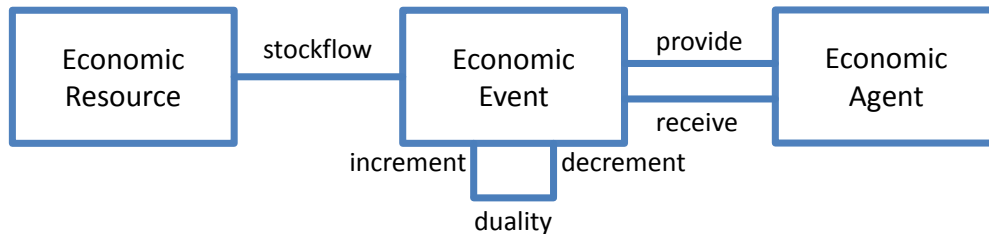


Figure 3.1: REA Core Concepts

The acronym REA stands for the three main concepts **resource**, **event**, and **agent** as depicted in Figure 3.1. In literature, they are often referred to as economic resources, economic events, and economic agents. Due to better readability of this thesis, we often drop the prefix term *economic*. Agents are persons, companies, or organizational units capable of having control over resources, who participate in an economic exchange. Resources are exchanged between the agents during an economic exchange. They can be goods, rights, or services which agents have control of and which need to be monitored or controlled in a business application. Economic exchanges consist of two to many economic events, where agents (inside and outside of the firm) exchange resources. An event is considered as a class of phenomena reflecting changes in scarce means [Yu76]. Exchanges can either be a transfer or a transformation. Transfers generate value due to an exchange with outside agents (e.g., selling a fish to a customer), while transformations generate value due to the change of a resource in form or substance (e.g., cleaning a fish).

There are three types of relationships: **stock-flow**, **participation** (provide and receive), and **duality** relationships. Stock-flow relationships link resources with events while participation relationships link agents with events. Another important concept of REA is the duality relationship. In general, during a transfer an inside agent gives up the control of resources to an outside agent. In return, the inside agent gains control of other resources from the outside agent. Thus, a

decrement event decrementing resources always has to be in a duality relationship with at least one increment event incrementing resources. This "give-take" constellation is a main principal of economic exchanges. For example, in a purchase (increment event) a purchaser (inside agent) may buy a fish (increment resource) from a fisherman (outside agent) and pay cash (decrement resource) during a payment (decrement event). As for transformations, we may consider an uncleaned fish (decrement resource) being cleaned by a cleaner (inside agent) in a clean-in event and the cleaned fish (increment resource) being received by a shop assistant (inside agent) in a clean-out event. Thus, the value of the fish is increased. This fish cleaning transformation example is a compromise, where the increment and decrement are conceptually congruent. Another transformation example is, where computer hardware like hard disk and processor (decrement resource) are assembled to a computer (increment resource).

Additional constraints on the REA language to restrict the use of the concepts are stated in three axioms [GM00b]:

- **Axiom 1:** At least one take event and one give event exist for each resource (guarantees modeling of economic activities as a sequence of exchanges).
- **Axiom 2:** All events effecting a resource decrement must be eventually paired in duality relationships with events effecting an increment and vice versa (ensures correct enumerations of exchanges).
- **Axiom 3:** Each exchange (transfer) needs an instance of both the "inside" and the "outside" agents (ensures presence of exchanges between parties with competing economic interests).

In [GM99] Geerts and McCarthy explain the benefits of using REA for analyzing, designing, implementing, and operating enterprise information systems and coupling it with a knowledge-based decision support system. Other than traditional business information systems relying on bookkeeping, REA allows for the semantic integration of data from different sources and thus, allows for better use by intelligent systems.

As mentioned above, REA evolved over the years and additional concepts have been introduced. In [GM00b, GM02] extensions of the REA ontology, both, vertically and horizontally have been added. The vertical layers specify the REA concepts at a different level of granularity. The top layer shows the most abstract representation of economic exchanges, whereas the bottom layer shows the most detailed representation of the economic exchanges. On the other side, the horizontal layers show the economic exchanges at a different point in time (i.e., what has happened or is happening, what is planned, and what could possibly be). An overview of the various layers is given in Figure 3.2. The vertical layers are marked as v1, v2, and v3. The horizontal layers are marked as h1, h2, and h3. The core REA ontology we just discussed resides on the intersection of the (h1) operational layer and the (v2) process specification layer and was accordingly extended later on to the other layers as explained in the following.

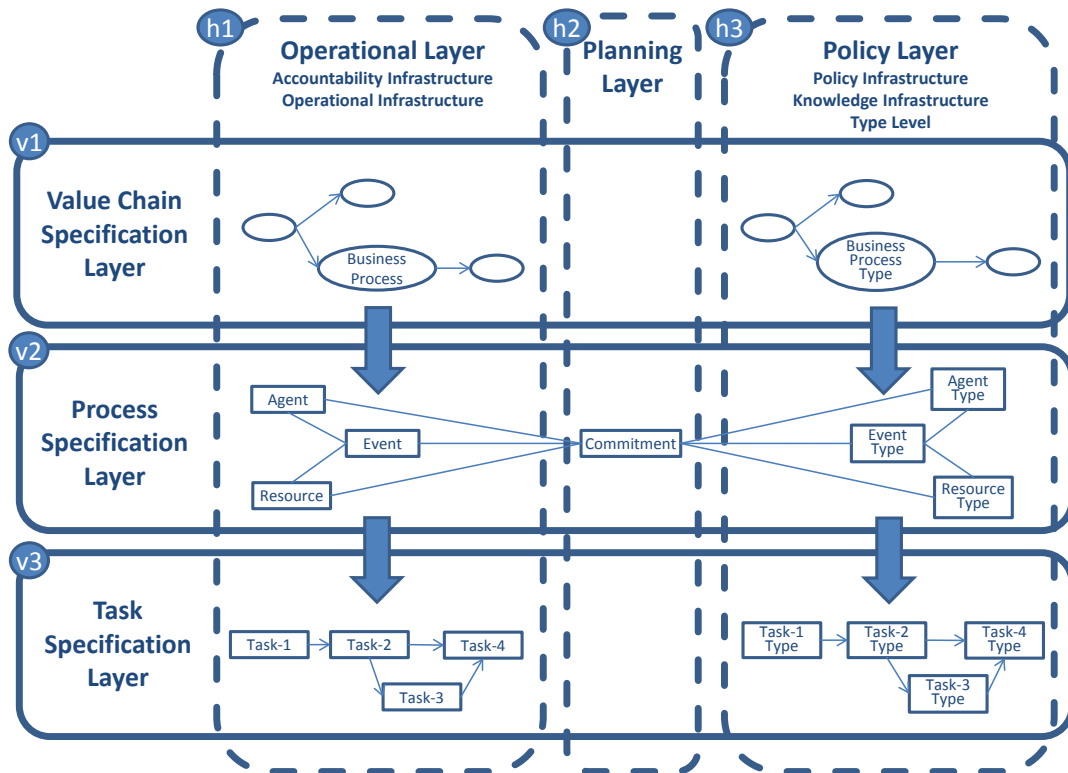


Figure 3.2: Vertical and Horizontal REA Layers [GM02] (extended)

3.1.1 Vertical Layers

Vertically, REA now consists of three different layers concerning entrepreneurial logic and workflow details at a different level of granularity. The three layers from top down are: the (v1) **value chain specification layer**, the (v2) **process specification layer**, and the (v3) **task specification layer**.

The process specification layer (v2) in the middle reflects the REA core concepts introduced before: resources, event, and agents as well as their relationships stock-flow, participation, and duality. It serves as the root for all the upcoming layers and therefore is the core of the REA ontology. Relationships to the other layers exist and thus, the core concepts can be referenced from the other layers.

The value chain specification layer (v1) resides above the process specification layer. According to McCarthy it is a purposeful chain of processes (conversions and exchanges) aimed at assembling the individual components of a final product (i.e., its portfolio of attributes) of value to the customer and in order to produce a firm-wide profit. Hammer et al. [HC93] de-

fine a process as "a collection of activities that takes one or more kinds of inputs and creates an output that is of value to the customer". Processes are connected with each other using resource flows. A resource given up by a process serves as an input to another process which gains this resource. The value chain is an abstract representation of the economic exchanges of the company. Thus, zooming into the process leads to the economic exchange defined on the core process specification layer (v2) describing the involved events, actors, and resources of the process. Consequently, resources flowing in and out the processes on the value chain specification layer (v1) correspond to the resource increments and decrements of the events on the process specification layer (v2).

Modeling business enterprises as value-added process hierarchies with REA is described in [GM97]. It mentions the drawbacks of traditional enterprise information systems based on non-semantic double-entry accounting models such as limited interoperability and reusability. Based on a semantic model, REA can tackle these limitations. The authors explain the usage of the REA ontology to model a value chain and dualities. Therefore, they base their example on a company called *Sy's Fish*, which is also subject to the accompanying example of this thesis.

The task specification layer (v3) is located beneath the process specification layer (v2), which splits the corresponding event from the upper layer into certain tasks. As elaborated in [GM01] a differentiation between events and tasks is domain dependent and highly heuristic. Tasks are for example used in cases where the completion of the task is not useful for making decisions (e.g., not for managerial, planning, designing, monitoring, recording, or evaluating purposes) or the task completion process highly depends on technology.

3.1.2 Horizontal Layers

Horizontally, there are three layers spanning over all three vertical layers describing exchanges at different points in time: the (h1) **operational layer** defining what events "are happening" or "have happened", (h2) **the planning layer** specifying what events "are planned" or "are committed" for the future, and the (h3) **policy layer** defining policies, standards, blue prints, and restrictions for events on a type level.

The operational layer (h1) (also called accountability infrastructure or operational infrastructure in literature) is normative and captures the events which actually occurred (the past and near present). Thus, the operational layer contains the basic REA concepts (events, resources, and agents as well as processes and tasks) outlined above and described in detail in the original REA paper [McC82].

The planning layer (h2), which is informative, is situated between the operational layer and the policy layer and defines commitments. Originally, commitments were proposed on the operational layer in [GM00b]. However, since that time McCarthy more often used in slides and discussions with us a separate layer for the commitments called the planning layer. Thus, we also use this additional layer in our thesis. Commitments reflect what is planned or scheduled (the scheduled future) and also specify relationships to concepts of the two other layers. The

notion of commitments in REA is based on Ijiri's definition [Iji75] who sees a commitment as an "agreement to execute an event in a well-defined future that will result in either an increase of resources or a decrease of resources". Accordingly, there is a fulfillment relationship between commitments and events in the REA ontology, meaning that a commitment should be fulfilled by an event in the future. Similar to the duality relationship between corresponding events, there exists a reciprocal relationship between corresponding commitments. This means, that a decrement commitment should be paired with at least one increment commitment. Commitments are grouped together either by contracts or by schedules. Contracts are executed in the future by transfers while schedules are executed by transformations. For example, a fish sale (transfer) is executed after an order (contract) was made by the customer or a fish is cleaned (transformation) by an agent after he was assigned to do so (schedule). Last, contracts and schedules can also be bundled by an agreement.

The policy layer (h3) (also called policy infrastructure, knowledge infrastructure, or configuration in literature) is informative and spans over all three vertical layers as well. It captures the abstract nature of real economic phenomena on a type level, and it additionally allows the definition of policies between various REA concepts. The main concepts introduced on this layer are type images which group actual phenomena having common properties together. The type images are resource types, event types, and agent types. An agent can be typified by an agent type, an event by an event type, and a resource by a resource type. Furthermore, policy relationships can be defined between the different previous concepts (e.g., events, agent types, resources, ...) specifying certain practices, standards, or constraints (e.g., which agent is allowed to participate in which event type, or which resource type costs how much in what event type).

We will now explain the concept of types by an example depicted in Figure 3.3. On the left side are the REA concepts from the operational layer and on the right side the REA concepts – types – of the policy layer. There are three levels from the bottom to the top according to MOF: The instance/database level on M0, the REA model level M1, and the REA meta-model level M2.

We are now looking at the M0 instance/database level. We have five different instances of an employee in the employee table, which are real-world employees: Steven, John, Eric, Michael, and Lisa. Additionally, we have two different employee types in the employee type table: cashier and salesman. Steven, John, and Eric are of employee type cashier while Michael and Lisa are of employee type salesman.

We are now looking at the M1 REA model (cf. the middle layer of Figure 3.3) which describes the concepts and their relations of M0 explained above. This is actually the modeling layer, where business experts and IT-experts will define the REA model of the economic phenomena of a specific company. For this example we assume that we have three different kind of agents in the company: customer, supplier, and employee. We have previously instantiated five employees on M0. Therefore, we define the three different REA agents customer, supplier, and employee, which will store the attributes specific to each single instance of these agents. Such an attribute is for example *name* or *date of birth*. Furthermore, we also define an agent type for each of these three agents: customer type, supplier type, and employee type. We have previously instantiated two employees types on M0. These types

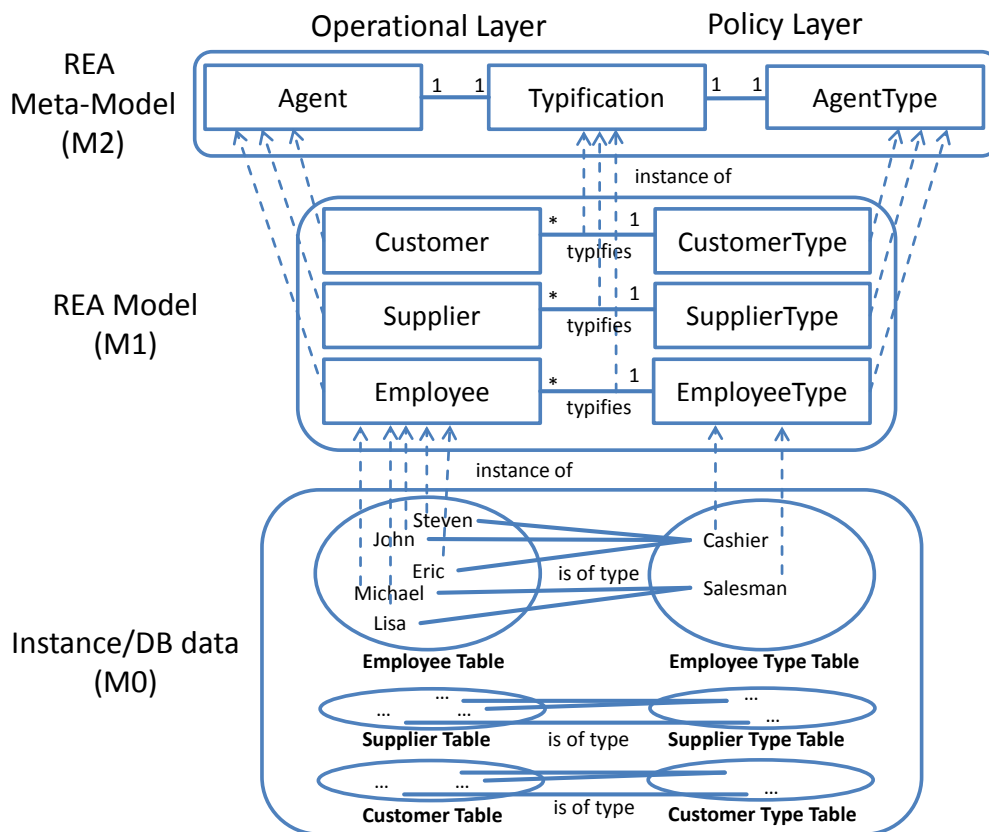


Figure 3.3: Type Model Layers

now contain attributes which are common to all agents of this type (e.g., starting wage for the employee type). The typifies relationship between for example `employee` and `employee type` now specifies, that on the instance level (M0) – the real world objects – there might be multiple instances of `employee` which are of the same `employee type` instance.

The M2 REA meta-model at the top of Figure 3.3 defines the concepts and their relations of the M1 REA model and specifies, that one agent (1) has exactly one *typification* (6) relationship to exactly one *agent type* (2) and vice versa. In the end this means that there is a one-to-one relationship between the meta classes *agent* and *agent type*. To better understand this relationship we compare this constellation with a concept from the object-oriented paradigm: Classes in the object-oriented paradigm may comprise static attributes (class attributes) and non-static (regular) attributes. In the REA context, it is required to have one concept covering the regular attributes and another one covering the static attributes. For example `employee` is an *agent* having regular attributes (e.g., employee id, name), whereas `employee type` is an *agent type* covering all static attributes that are common to all `employees` (e.g., minimum starting wage). The separation of these two concepts is especially important for the relational database system as well as for defining policies.

The type concept described above is also specified the same way for resources and events. The example was based on the process specification layer, but also applies to the value chain specification layer in terms of process types and to the task specification layer in terms of task types.

To add constraints or possible configurations to types and other REA concepts, relationships may be added between these concepts. In [GM05] two different types of relationships are mentioned: policies and standards. Policies restrict the possible configuration of actual types and REA objects. For example, only certain types of employees are allowed to sell fish to restaurant customers. These restrictions can be taken to validate actual phenomena. On the other hand, standards do not restrict but serve as a blueprint. For example, one may define how much a fish costs for a sale in a specific area. Another example would be, how much time is usually used to clean a certain amount of fish and what resources are usually used to clean the fish. If an event type associates another type (i.e., agent type or resource type), then this association is called "specifies" (e.g., for an amazon expedition we define that we need a guide, but we do not define which guide it has to be exactly). On the other hand, if an event type associates a specific agent or resource, then this association is called "reserves" (e.g., for the amazon expedition, only John can be the guide).

In [GM06] Geerts and McCarthy introduce additional concepts of the policy layer and explain existing concepts in more detail. The newly introduced concepts are generalization and grouping. Taking the example from above, we explain the idea of generalization. An agent can be any of the three different agent types: cashier, storekeeper, and salesperson. It is assumed that they all have the same common property starting-wage, whose value varies between the three different type instances. Therefore, we can have a generalized agent called employee agent containing the common starting-wage property and the three agent subtypes cashier, storekeeper, and salesperson containing individual properties such as sales experience. Another concept introduced is grouping. It describes the membership of heterogeneous objects in a collection. For example, an employee may be a member of the group external employees or a member of the group internal employees. Thus, policies can be applied on a group of agents instead of specifying them on each single agent. Furthermore, [GM06] introduces five different patterns for policy layer specification: basic, mirror, compromise, hybrid, and root. The patterns specify different ways of restrictions between and within the policy and operational layer.

Two different natures of resources are discussed in [GM04, GM06], called the car and nail resources. Car resources are resources, where each individual resource has to be traced. This might for example apply to the resource equipment sold in a store, where each item is tagged with RFID. Nail, on the other hand, is a resource which does not have to be traced individually. An example might be the resource fish, where the tracking of each individual fish is not needed, but where just the quantity on hand is recorded. In this thesis we came up with the name *bulk resource* for these nail resources, because we believe that the term "bulk" better reflects the meaning of these special resources.

3.2 REA Representation and Formalization Approaches

REA still lacks a thorough formalization defining exact rules and constraints for REA providing a common understanding of REA for all parties. Additionally, a dedicated representation for REA helps to understand and model the REA ontology. Therefore, we now discuss various attempts of representation and formalization approaches for REA tackling these problems.

In the first working draft of the REA specialization module [UN/08] for the UN/CEFACT's Modeling Methodology (UMM) [UN/10] attempts were made to formalize REA as an UML Profile (cf. Figure 3.4). We adapted the figure by changing the composition between *EconExchange* and *ExchangePhase/EconResource/EconAgent* to an aggregation, because one and the same *ExchangePhase/EconResource/EconAgent* can be in multiple exchanges. UMM is a standardized methodology for modeling the global choreography of inter-organizational business processes. A UML Profile tailors existing UML (Unified Modeling Language) [Obj11c] languages to a specific domain. Besides formalizing the concepts of the REA process specification layer already described, some more additional concepts are introduced. Among those are business location, economic claim, and exchange phase. A business location captures the site, where an economic event takes place and thus, might be important in terms of legal aspects. An economic claim indicates a temporal imbalance in a duality (i.e., two events in a duality are not executed at the same period of time). Last, an exchange phase describes the related phase of an economic exchange [Sch10] according to the Open-edi business transaction phases [ISO11b].

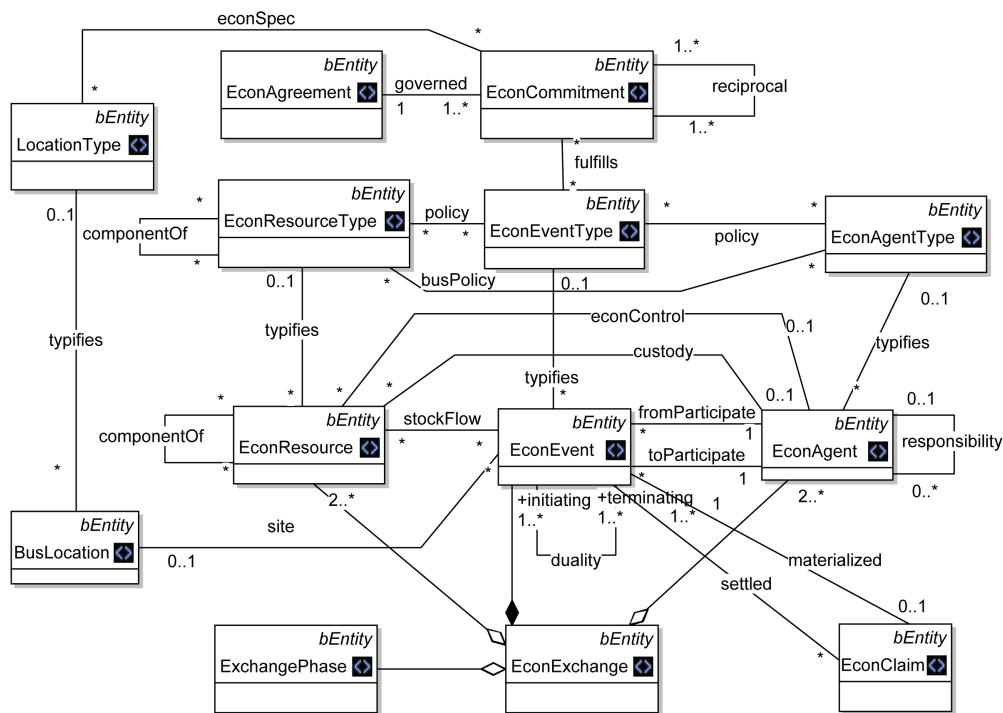


Figure 3.4: REA UML Profile [UN/08, Sch10]

Reengineering REA to create a new conceptual representation is proposed in [GP07a, GLP08]. Gailly et al. criticize, that the current representation of REA leaves space for diverging interpretations of the ontology [GLP07]. Therefore, they follow a three-phased reengineering methodology based on the METHONTOLOGY framework [GR99] for an improved REA representation. In the first phase called Reverse Engineering, the conceptualization of REA based on the different current representations and interpretations was acquired. In the second phase called Restructuring Phase an improved conceptual representation of REA was created as an UML (Unified Modeling Language) class diagram. In the third phase called Forward Engineering the new representation is mapped into a formal representation based on OWL [GP07b]. A REA model of an online bookshop is provided using UML class diagram artifacts as an example of an instantiation of the new conceptualization. The authors focus on creating an improved REA conceptualization but do not make use of a dedicated graphical representation. Hence, it is hard to comprehend the REA models, and it is cumbersome for non-experts to create a REA model. In our work, we want to overcome this limitation by using domain-specific languages. Accordingly, we also follow a different methodological approach focusing on the development of domain-specific languages by Strembeck and Zdun [SZ09].

Similar to the just introduced OWL approach in [GP07b], Zhang et al. [ZJWL10] propose an OWL two-layer enterprise information architecture based on REA. These two layers are the strategy layer and the operational layer. First, the strategy layer incorporates concepts from the REA value chain (which are not considered in [GP07b]) and describes the internal and external macro-environment of the enterprise. Second, the operational layer is based on the REA core-concepts including typification (i.e., resource, event, agent, and their corresponding types) and captures the enterprise business activities. With their approach the authors want to foster the reuse of business models and make use of REA in knowledge-based management systems. Since XSD schemas are easy to understand and broader adopted than OWL, we believe that in our work we can achieve better interoperability between systems and therefore, gain better acceptance by the community.

In [Gai10], a UML Profile for REA is introduced. A UML Profile can be used to assign a certain domain to specific UML diagram elements. Thus, the general-purpose language UML can be adopted to a domain-specific language. Twenty stereotypes representing the REA concepts were included in the REA profile. Additionally, to cover the REA axioms, OCL (object constraint language) constraints were used to limit the misuse of the REA UML Profile. Once again, this approach still does not provide a dedicated graphical language, which in our opinion does not foster the comprehensiveness of the REA ontology compared to our proposed REA-DSL. In our REA-DSL implementation, instead of using OCL constraints, we create rules attached to the DSL. These rules contain procedural code assuring that the REA model adheres to the REA axioms. These rules are checked during modeling.

Different views on REA are given in [CPKK06] by using feature models. The idea is to find relations between feature models and ontologies (in this case REA) and provide different views on it. In the provided example, a business feature model and an administration feature model were created. The view of the business feature model contains customer-concerning events (e.g., Checkout, Review), whereas the view of the administration feature model contains system administration concerns (e.g., BehaviourTracking). The views provide smaller subsets of the

model instead of seeing the whole model. In our work, we already reduce the complexity of REA beforehand by providing separate duality sub-models and using the different REA layers.

Some efforts to develop a new version of the REA ontology have been taken by the REA version 2 wikipedia page [Man11]. However, since the end of 2007 there has not been a lot of activity on this wiki and the information on the page is sparse.

Sedbrook [Sed12] developed a first draft of a domain-specific language for REA policies. Similar to our work, he bases his approach on the Microsoft Visualization and Modeling SDK. However, a thorough formalization by a comprehensive meta-model is missing. Policies are described mainly by validation rules applied on certain properties of REA objects. In our approach to incorporate policies in the DSL, we focus on providing data structures to record policies between different REA concepts such as which resource type may be exchanged by what agent types in what event types. However, extending and integrating the validation rules by Sedbrook [Sed12] might be valuable for future work.

Laurier and Poels [LP07, Lau10] describe three fundamental patterns underlying the REA business model ontology. They argue, that they can be used as building blocks for a universal modeling template for business processes and can use these fundamental patterns to create more complex patterns. Furthermore, these patterns may be used for describing supply chains or value chains.

3.3 REA in Accounting Information Systems

Even before his initial REA paper from 1982, McCarthy started merging the ideas of accounting theory with database systems without relying on a specific system [McC79, McC80]. He mentions, that an accounting system is most naturally modeled in a database environment as a collection of real world entities and relationships among those entities. Therefore, he uses an Entity-Relationship (ER) diagram to model an example retail enterprise. Another work of McCarthy together with Gal [GM86] concerns the implementation and operation of a relational accounting system to demonstrate the simplicity of relational procedures and their usage to create events accounting systems.

The automated integration of enterprise accounting models throughout the system development life cycle is discussed in [GMR96]. In regards to the five systems development life cycles (SDLC), various computer-aided software engineering (CASE) tools based on the REA ontology are presented in [GMR96]. Among those are CASE tools such as REAVIEWS, CREASY, and REAtool [CMO94] as well as the tools REACH and FREACC which were still in development in 1996. Best to our knowledge, we did not find any evidence that these tools are still supported/used today. The goal of REACH [RM99] for accounting system design is to help modeling an accounting database system in conjunction with three different accounting knowledge types. These types include (i) using REA as a first-order domain theory, (ii) providing reconstructive expertise from textbooks in the accounting domain, and (iii) utilizing implementation heuristics derived from the design experience of database designers. With this approach, the authors want to overcome the limited capabilities of computerized ledger implementations. Geerts et al. mention, that the REA-associated data components form only a part of the data needed to operate and manage an enterprise well. In our work we create a tool using the dedicated REA-DSL notation

based on a formalized REA ontology extended with database concepts in order to fully derive relational tables for accounting information systems. We argue that using this special language fosters the use and acceptance of the tool.

In [Poe03, PMG⁺04] Poels et al. compares REA with Entity-Relationship (ER) diagrams in terms of conceptual modeling of accounting information systems. This empirical study focuses on the perspective of the users who need to use and understand the conceptual REA models (e.g., database designers, system developers, information analysts, and accounting information systems (AIS) end-users). The participants in the study were business administration students enrolled in a post-graduate AIS course. The authors proved, that the accuracy of comprehension of the REA diagrams compared to ER diagrams was significantly higher (5% significance level). The hypothesis, that understanding REA diagrams takes less time than understanding equivalent ER diagrams could be proven on the 10% significance level. Thus, the authors showed the effectiveness of REA modeling in the area of AIS. Furthermore, since current REA notations are closely related to the ER notation, we expect an even higher comprehension of our REA-DSL compared to the current REA ER-like notation.

Dunn et al. [DM97] elaborate on the past research work in accounting information systems and clarify the differences between the terms events accounting, database accounting, semantically-modeled accounting, and REA accounting. A comparison between the REA model and the SAP data model to find the relationships between them is conducted by [O'L04].

3.4 REA Extensions and Integrations

In this section we dwell on literature concerning the combination of REA with different frameworks and concepts as well as proposed extensions and changes to the current REA ontology.

The lack of knowledge reuse and knowledge sharing between existing accounting systems slows down the design and implementation of new accounting systems [GM00a]. A common semantic infrastructure based on REA is proposed to overcome this obstacle. Using this common semantic infrastructure, accounting concepts can be shared across functional boundaries and be used in different applications/systems. The enterprise data is annotated with REA ontology concepts to achieve this. The conceptual structures can then be used for augmented intensional reasoning, which refers to the active use of conceptual structures in information systems operations. This may lead to reduced costs and faster design and implementation of accounting information systems. Since we also base our REA-DSL on the REA concepts, users can also benefit from higher interoperability.

An ontology-based domain-driven design for software applications is introduced in [Hru05]. The application objects are based on the REA ontology and are extended by so called application aspects. These application aspects are identified by the user requirements. The term aspect is well known from aspect-oriented programming, where different functionalities like logging or transactions are seen as different cross-cutting aspects. The author uses this term, because the user requirements cross-cut the application objects. Using the REA ontology as a semantic basis for the objects of the application (object dimension) and adding the user requirements as aspects (aspect dimension) creates an application object model based on strong economic concepts. In comparison to our work, this approach tries to tie requirements into the existing REA

notation based on class diagrams, whereas we provide a new domain-specific representation. One interesting idea in this paper is capturing the requirements of the specific business domain of a company by means of wizards and then automatically creating corresponding REA-DSL models. However, this idea is out of scope for this thesis and is interesting for future work.

The use of REA (and e³value) as part of the early requirements modeling is discussed in [GEPP08]. Instead of using REA as a domain-specific language, a REA profile for the i* modeling framework [CNYM00] (which is used in the early phases of system modeling) is introduced. The ontological correctness of i* models in the business domain is enhanced by combining them with the REA ontology and thus adhering to specific guidelines. We believe, that our REA-DSL usually can be used in the requirements elicitation phase as an easy to understand language without the integration into an additional requirements modeling framework.

The value chain of the current REA ontology can only depict flows of resources at the operational layer. In [HHKV10] Hunka et al. apply the REA value chain on a production planning model, thereby integrating activities such as planning, controlling, and monitoring which reside on the policy layer into the value chain. Thus, the value chain model of REA additionally captures flows between the operational and the policy layer.

In [HHKV09] the authors discuss a different approach of using REA commitments. They see commitments on a sublayer (we call it planning layer) between the operational layer and the policy layer. By adding a grouping relationship between event types and commitments, they want to raise the commitments to the policy layer. Commitments can be composed of multiple committed elements, which replace the many-to-many fulfillment relationship between events and commitments.

Weigand et al. [WJA⁺11] use REA as the basis of a value-driven framework for service management in a SOA (Service-Oriented Architecture) environment. The service management framework introduced is the glue between business services and software services. It is based on REA extended by a management ontology and follows a business-driven and service-oriented approach. Policies can be enforced on the operational services using the proposed framework.

By building our formalized REA-DSL on a meta-model and using XML as the storage and exchange language, we hope to support and foster further approaches to incorporate REA with other areas and provide a stable base to add extensions.

3.5 REA Applications and REA Mappings

In this section we dwell on work which uses and applies REA as well as work considering mappings to other modeling languages. Access Control Policies and their validation in conjunction with REA is presented in [KC09]. The authors introduce their own policy artifacts and do not make use of the REA policy infrastructure. In our REA-DSL, a basic policy infrastructure is included and we believe, that such validations can be performed on our REA-DSL right away. [MM02] have done first steps to build a school information system for a real school upon the REA concepts. First, REA is used to model processes such as registration. In a second step, the REA models are used as a basis for the creation of a database system. A Revenue Information System (RIS) is developed in [RAM09] using the REA ontology in order to evaluate it in a real world case study.

Mappings between REA models and other business and business process models are introduced in [MS09, SMHW10]. The commonalities between REA and e^3 value are elaborated on in [MS09]. Furthermore, a mapping between these two languages is provided. e^3 value primarily focuses on the profitability of the IT system but lacks accounting sophistication. REA concentrates on implementation and alignment issues of an IT system and is also concerned about the profitability. There are some concepts between the two languages. The REA core concepts (i) resource, (ii) event, and (iii) agent are mapped in the provided order to the e^3 value concepts of (i) value object, (ii) value transfer, and (iii) actor. Another mapping is provided by [SMHW10] between REA and UMM. UMM is a standardized methodology to model global choreographies of inter-organizational business processes. By discovering the connection points between REA and UMM, the UMM business processes can be aligned to the domain rules provided by REA ensuring the economic reciprocity. Thus, UMM model stubs can semi-automatically be derived from REA models. We argue, that the mappings provided can be used between our REA-DSL and DSLs of e^3 value and UMM. The two mappings discussed provide mappings to the core concepts of REA. Our proposed REA-DSL contains a more comprehensive set of concepts going beyond the core concepts of REA used in these mappings. These extended concepts might be subject to derive additional artifacts to/from e^3 value and UMM not yet provided by the discussed mappings.

REA value chain modeling is used as an alternative to business process modeling by Vymetal [VHHK10]. A transition from REA to UML activity diagrams is provided. Furthermore, UML sequence diagrams and UML state diagrams are provided for the REA model to offer a different view.

Interoperability between different e-collaboration modeling standards is discussed in [GP09]. The two languages used in this study for e-collaboration modeling are UMM and ISO/IEC 15944 [ISO07]. The authors argue, that both business partners who want to interact on the business process layer, have to adhere to the same business process modeling language in order to achieve interoperability. To overcome this limitation, the authors propose to use REA as a shared semantic ontology basis. Hence, UMM and ISO/IEC 15944 can be transformed to each other by using the intermediate REA model. This allows business partners to use different business process modeling standards for their e-collaborations.

REA by Example - Sy's Fish Wholesaler

In this chapter we introduce the accompanying example of this thesis. It is based on a real company. The company is called Sy's Fish which acts as a fish wholesaler on the market. This example was also used in one paper of Geert and McCarthy concerning modeling business enterprises as value-added process hierarchies with Resource-Event-Agent object templates [GM97]. We have adapted the example in order to demonstrate the full power, i.e. all concepts, of the REA-DSL. However, to keep the example easy to follow, possible phenomenas such as advertising, rent expenditures, or tax payments are not considered. Nevertheless, these economic activities can in fact be described by the REA ontology. In the following we will explain the main business idea behind the company and represent it with the former REA ontology modeled with class diagrams.

4.1 The Business Idea

Sy's Fish is a distributor of seafood and provides his base of restaurant customers and private customers with over 50 types of fish which can be stored at all local stores. Additionally, he sells books about cooking fish and fishing rods to his customers. An overview of Sy's Fish business idea and essential activities is shown in Figure 4.1 and marked by small circled numbers. Sy's Fish (1) purchases fish from various fish markets (2). Additionally, he also buys fishing rods and books about cooking fish from the factory (3). In order to transport the fish from the fish market to his stores and to deliver the fish to the restaurant customers, he keeps a fleet of trucks (4). The trucks are leased on yearly contracts, and lease payments are made monthly. Before the fish are sold to restaurants or private customers, they are cleaned within the stores (5). To perform all activities, Sy's Fish employs employees (6) with different qualifications. They fill out time cards fortnightly upon which they may note the percentage of time devoted each day to buying, cleaning, and selling fish. An employees can be a salesman, shop assistant, or cashier.

All the activities explained so far requires Sy's Fish to invest money. In order to make money, Sy's Fish has two main distribution channels. First, he sells the fish, fishing rods, and books to private customers (7) in his store. Second, he sells fish in high quantities (and also fishing rods and books if customers request so) to restaurant customers (8). Customers can pay with cash. Cash receipts and disbursements are made to/from one of the multiple checking accounts of the firm.

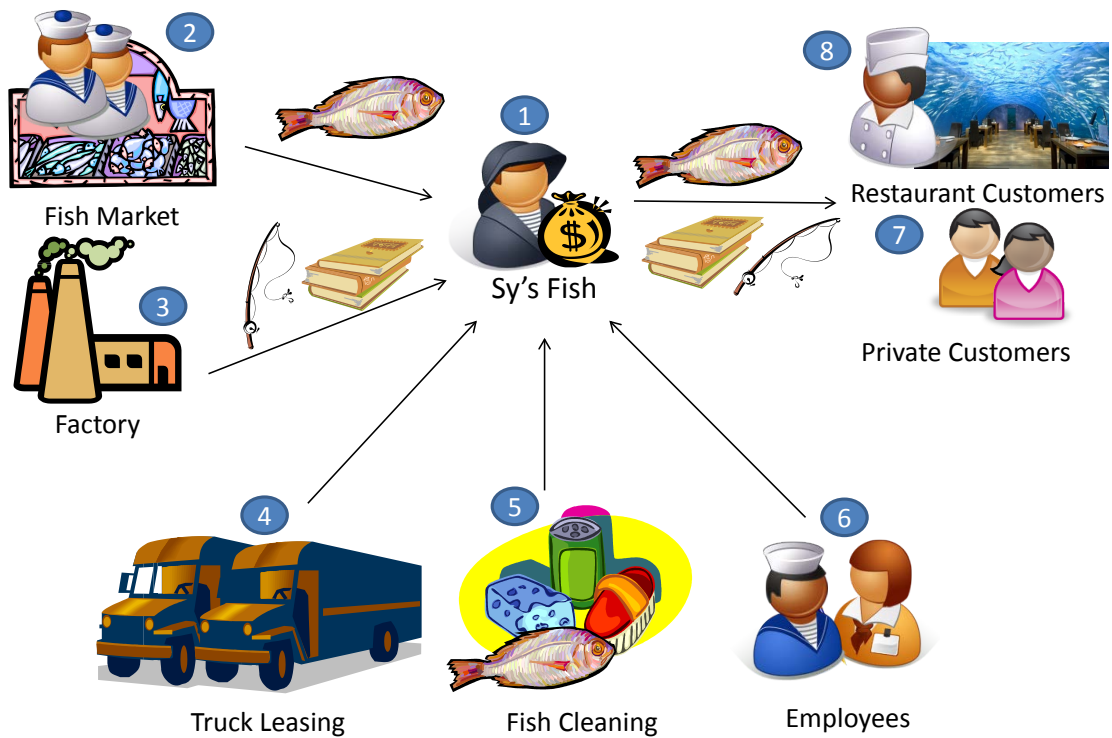


Figure 4.1: Sy's Fish Overview

This overview gives a good idea, how Sy's Fish makes money - the entrepreneurial script. We can identify following processes for the value chain (i.e., a chain of processes increasing the value of the product for the customers and leading to a profit for the company): payroll, fish buying, product buying, truck acquisition, transport of products and fish, cleaning fish, and selling the fish and products to private and restaurant customers.

4.2 REA Class Diagram Representation

In Section 3.1 we have described the core concepts of the REA ontology. We now apply the REA ontology on Sy's Fish company introduced before and represent it as a class diagram. At present, a class diagram is the usual representation for REA ontology models. In Figure 4.2 the REA meta-model is depicted. As a first step, we create a REA model incorporating the

basic REA concepts of the operational layer shown on the left side of the REA meta-model, i.e., *resource*, *event*, and *agent*.

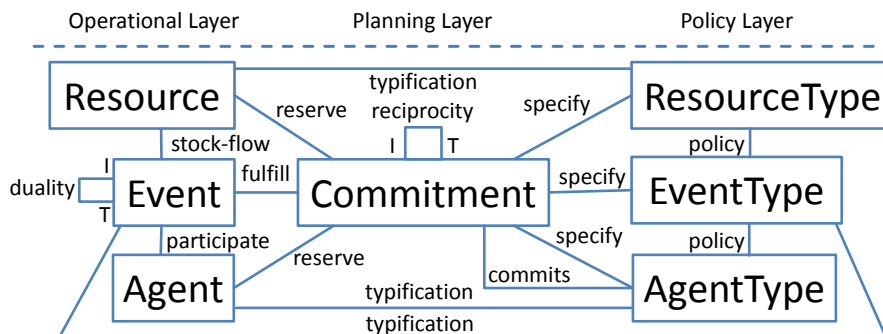


Figure 4.2: REA Meta-Model

We start by modeling the REA object constellation for the restaurant fish sale depicted in Figure 4.3. On the left side you see an instance example of a *sale* on the *12th February 2012*. The inside agent *salesperson Sy* sells *20 pounds of the fish carp* to the outside agent *customer McFish*. According to the duality principal of economic exchanges, the *customer McFish* pays *100 dollars cash* to inside agent *cashier Lisa*. On the right side of Figure 4.3 you see the corresponding REA object constellation using the class diagram notation showing the general aspects of a sale. The exchange consists of two economic events: *fish sale* and *payment*. They are connected by a duality relationship, where the *fish sale* event is the initiating event and the *payment* is the terminating event. In a duality we always have to have an event incrementing resources and an event decrementing resources. In this case, the *fish sale* event (annotated by “-GIVE”) decrements the resource *fish*, which is connected to the fish sale event by a *stock-flow* relationship. The resource *fish* is provided by the inside agent *salesperson*, who gives up the resource. The outside agent *customer* receives the *fish*. The agents are associated with the event with participation relationships. On the other side, we have the event *payment* (annotated by “+TAKE”) incrementing the resource *cash*. The resource *cash* is provided by the agent *customer* and received by the agent *cashier*.

In Figure 4.3 we have now shown an object constellation for the core REA concepts on the *operational layer*. REA also includes concepts for commitments and types residing on the plan and policy layer of the REA meta-model (cf. right side of Figure 4.2). Commitments model orders and define who commits to these orders.

Such an order – in this case ordering a fish – is depicted in Figure 4.4. On the left side you see an instance example of an order on the *10th February 2012* for the sale on the 12th February 2012 described before. We recognize that it looks similar to the fish sale discussed previously in Figure 4.3. This is because an order actually reflects, how the sale has to look in the future and by which events it has to be fulfilled. In our example, the *customer McFish* made an *order* on the *10th February 2012*. In this order, the inside agent *salesperson Sy* commits to sell *20 pounds of the fish carp* in the future to the *customer McFish*. In return, a future *payment of 100 dollars* is made by the outside agent *customer McFish* to the inside agent *cashier Lisa*. According to the

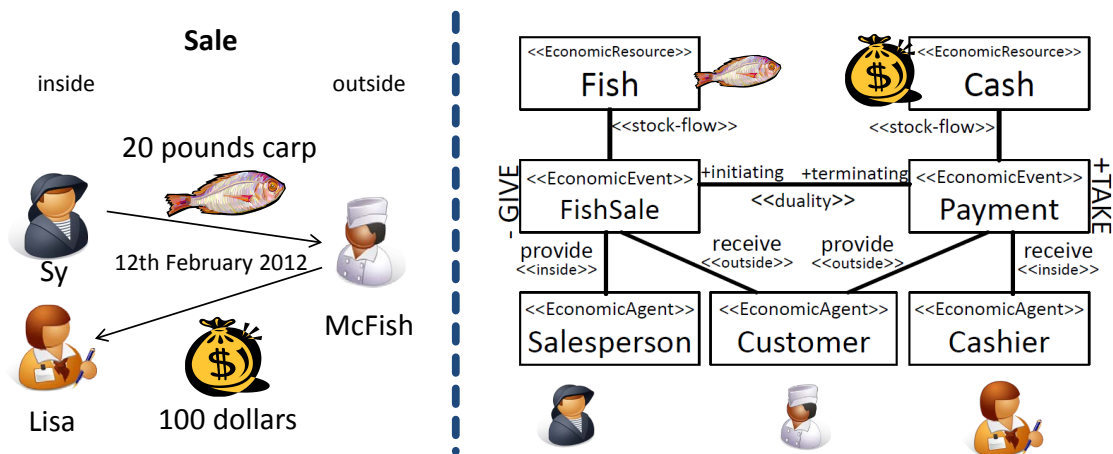


Figure 4.3: Sy's Fish Object Constellation for Restaurant Fish Sale

reciprocity relationship in the meta-model in Figure 4.2 a commitment specifying a resource-type decrement in the future has to be paired with a commitment specifying a resource-type increment in the future. As in our object constellation example on the right side showing the general aspects of an order, the *sale* commitment is in a *reciprocity* relationship with the *pay* commitment. The scheduled *sale* acts as the initiating commitment and the scheduled *payment* as the terminating commitment. In the *sale* commitment, the agent *salesperson* commits to sell a specified resource type *fish*. Additionally, the agents *salesperson* and *customer* are reserved for the future event. On the other side, in the *pay* commitment, the agent *customer* commits to provide the specified resource type *cash*. Additionally, the agent *customer* is reserved and the agent type *cashier* is specified for the future event. In general, there is a *reserve* relationship to a *resource/agent*, if the exact *resource/agent* is known at the time of the commitment and a *specify* relationship to an *agent type/resource type*, if the exact *resource/agent* is not known at the time of the commitment. The *fulfill* relationships between *sale* commitment and *fish sale* event as well as between *pay* commitment and *payment* event define, that these commitments are fulfilled by these events. The events are shown by a dashed class, because they actually relate to the events defined in the sale object constellation of Figure 4.3.

The meta-model in Figure 4.2 also defines policy relationships between agent types, event types, and resource types which are not depicted in the examples. With the policy concept, standards and restrictions for events can be defined. For example, we can define that for a *sale* event with *restaurant customers*, only experienced *salespersons* are allowed.

It is obvious how complex this representation might get modeling all aspects of Sy's Fish company. To get an idea, we provide an example of Sy's fish value chain on the operational layer including the value activities payroll, fish buying, product buying, truck acquisition, transport, cleaning, and selling. Each of these value activities consist of two events connected by a duality relationship. Figure 4.5 shows the object constellations for the value chain as an UML class diagram with a sequence of seven duality relationships according to the seven value activities mentioned above. Events from different dualities are connected to each other through a resource

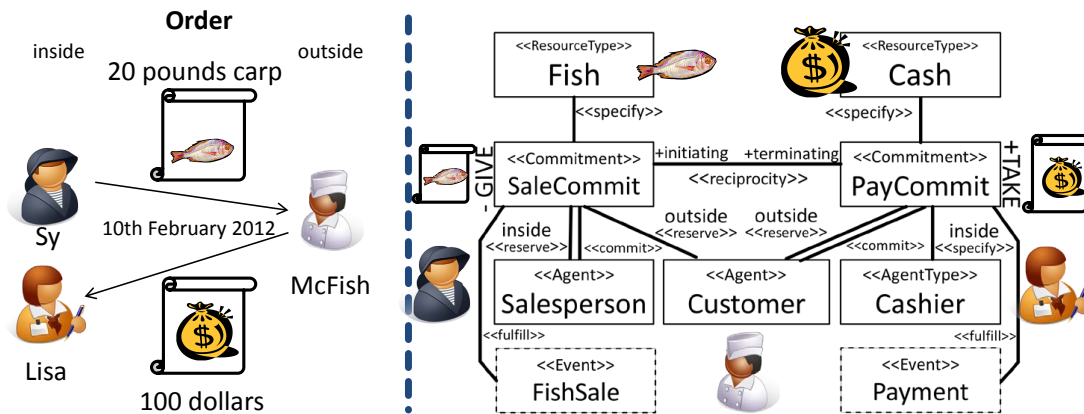


Figure 4.4: Sy's Fish Object Constellation for Restaurant Fish Order

and its resource flows. This means, that a resource serving as the output of one duality/value activity serves as an input for another duality/value activity. Each decrementing event "gives" away at least one resource and each incrementing event "takes" at least one resource. Furthermore, for each event at least one inside agent has to participate. In the case of a transfer, also at least one outside agent receiving the resource has to participate. For example, the truck acquired (take) from the truck acquire event in the truck acquisition value activity serves as the input resource (give) for the transport-decrease event in the transport value activity. We can see by this rather small example, how the complexity of the model increases and how it gets harder to comprehend the REA model, especially for non IT experts. Thus, the class representation is not an appropriate representation for business domain experts.

4.3 REA Ontology Limitations

A first limitation of REA is given with regard to the clarity of ontological statements. Although the REA model has evolved into a domain ontology, REA leaves space for diverging interpretations of the relationships between core concepts. In particular, the multiplicities for individual relationships are not clearly defined. Moreover, it is not clear if an event can be related to multiple stock-flows and agents. Furthermore, there is no axiom restricting event-resource (stock-flow) relationships.

In its original specification, REA allows the connection of a single event with stock-flows to resources which are incremented and at the same time to those which are decremented. However, this is not compliant with the duality concept which is established between an increment event(s) and an decrement event(s) which are well distinguished from each other. Accordingly increment events have to be connected by stock-flow relationships to resources which are incremented and decrement events to those which are decremented.

These ambiguities and potential semantic inconsistencies may be due to a lack of a dedicated specification language. In the current state, no detailed means have been proposed to specify precisely how to relate the REA concepts and how to enforce compliance of REA models with

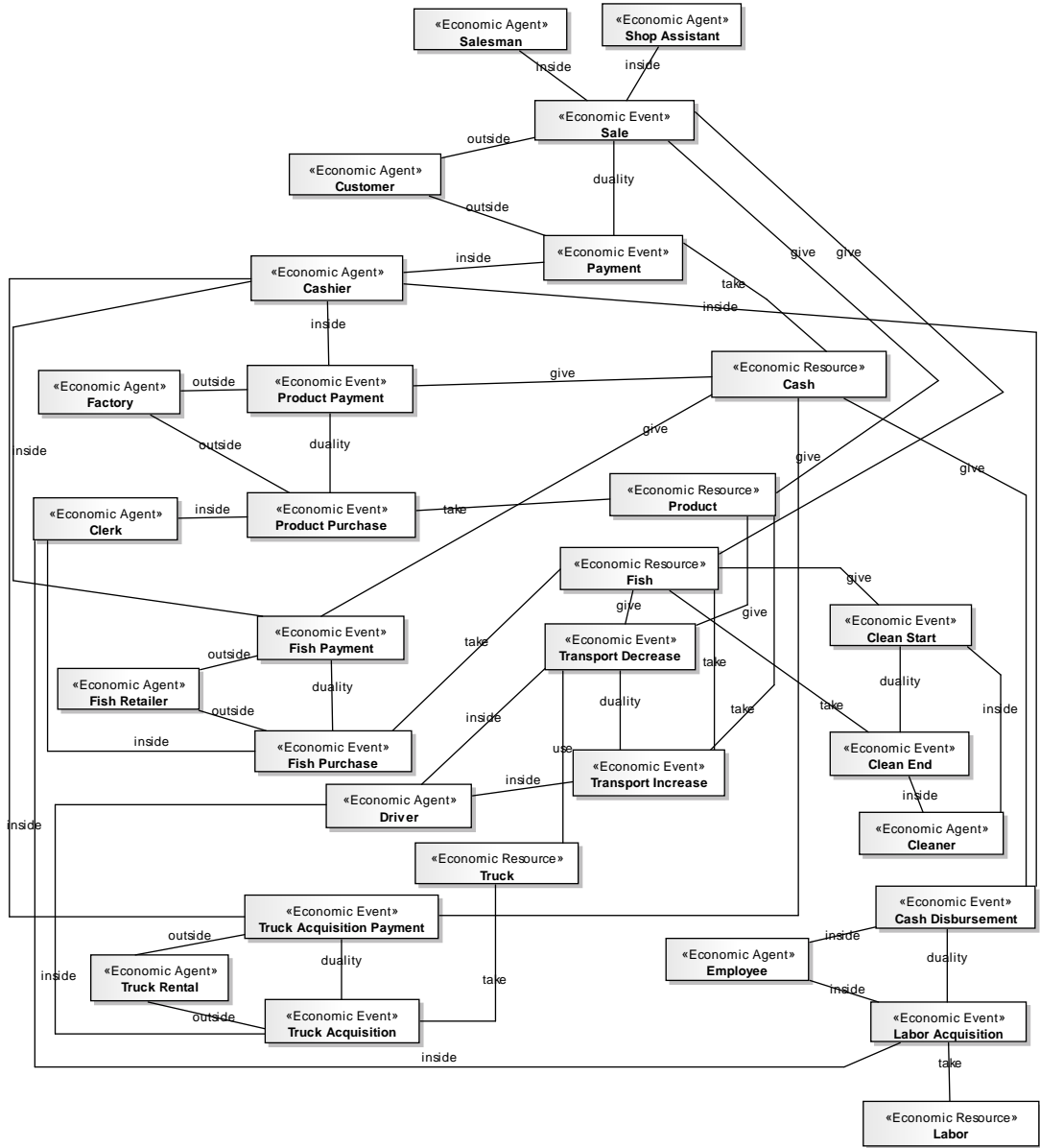


Figure 4.5: REA Sy's Fish Example

the axioms stated above. The compliance of REA models can only be checked manually. Thus, domain experts are responsible for incorporation of the REA pattern into particular enterprise information architectures. There is no conceptual facility that enables the development of interoperable REA models.

A third limitation is the complexity of REA models which increases progressively even with very few processes to be modeled. Each process generally includes at least eight entities which have to be modeled (two events, two resources, two inside agents and two outside agents) [GM99]. A dedicated REA modeling notation would help to overcome the complexity. However, proposing a modeling notation necessitates resolving the inconsistencies and incompleteness of the ontological statements.

REA-DSL

As mentioned previously, the current class-diagram-like representation of REA fosters diverging interpretations, and it is hard to understand. Consequently, we started to develop a graphical domain-specific language (DSL) for REA called the REA-DSL. We based the development of the REA-DSL on the Object Management Group's (OMG) four level meta-modeling architecture called Meta-Object Facility (MOF) [Obj11a] (cf. Figure 5.1).

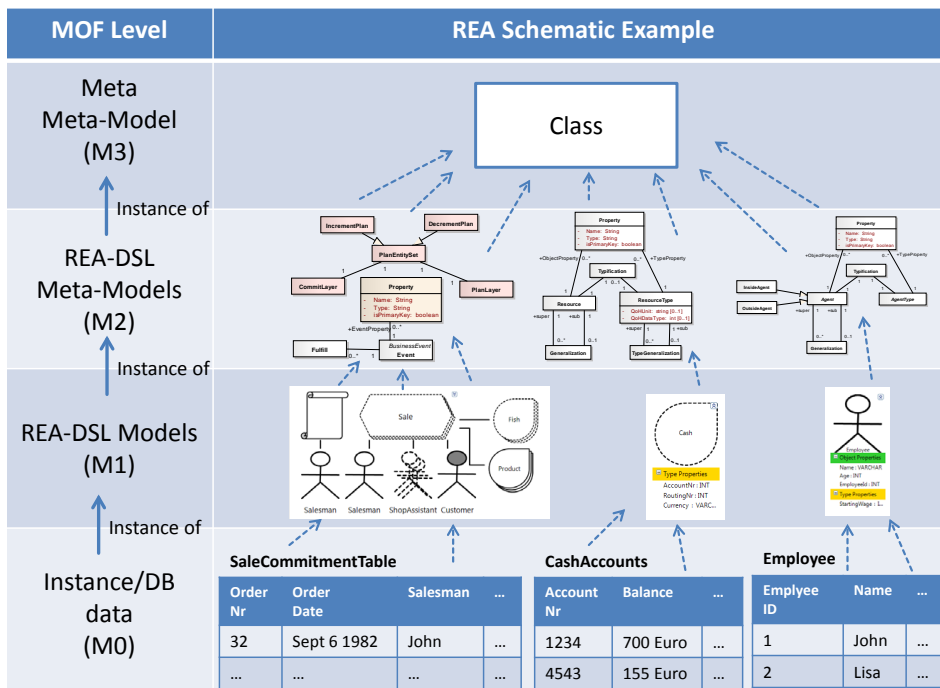


Figure 5.1: MOF Levels Applied on the REA-DSL

The four MOF levels are: the M3 meta-meta-model level, the M2 meta-model level, the M1 model level, and the M0 instance level. The meta-meta-model (M3) allows us to define the structure, i.e., the abstract syntax, of the REA-DSL as a meta-model (M2). The actual REA-DSL models are on the model level (M1). The recorded data in the database (e.g., for an actual sale) reside on the instance level (M0).

Accordingly, we define the structure for our REA-DSL on the M2 meta-model level. The REA-DSL meta-model comprises five interlinked views depicted in Figure 5.2 : (i) *resource view*, (ii) *agent view*, (iii) *value chain view*, (iv) *operational view*, and (v) *planning view*.

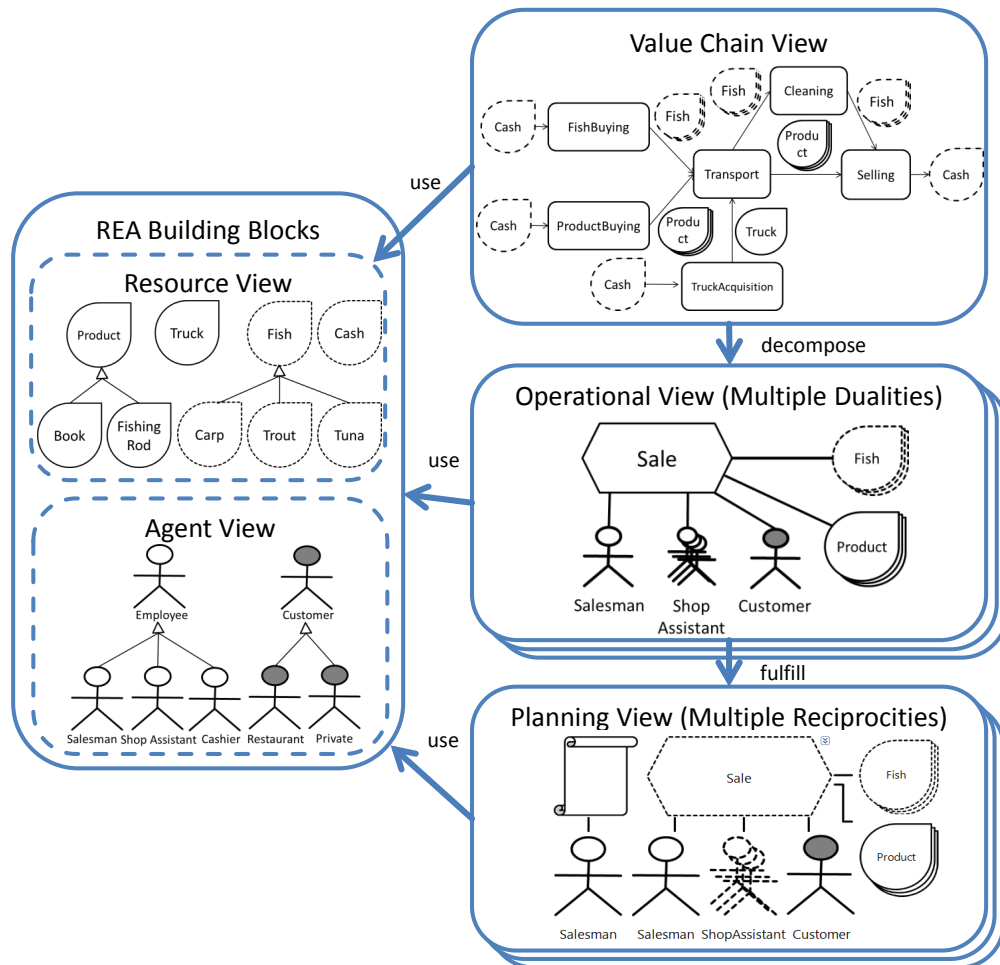


Figure 5.2: REA Views

Resource view (i) and agent view (ii) are the *REA building blocks*, thus, the resources and agents specified in these views are used by the three other views: value chain view, operational view, and planning view. The value chain view (iii) specifies the resource transfers and transformations of a company according to its entrepreneurial script in order to increase value. These value transfers and transformations are further specified in more detail in the operational view

(iv) where in multiple duality models events and their related agents and resources are depicted. Thus, the value chain view can be seen as an abstract representation of the underlying duality models in the operational view. The planning view (v) specifies in multiple reciprocity models the contracts and future events, which will get fulfilled by the events in the operational view. Notice, according to the REA layers introduced in Figure 3.2 in Chapter 3, the value chain view and the operational view reside on the operational layer, whereas the planning view resides on the planning layer. The building blocks agent view and resource view reside on all horizontal layers, because they can be used by either the operational layer, the planning layer, or the policy layer.

Before we dwell on the various views in detail, we give a brief introduction of the various notation elements of our REA-DSL in the subsequent section. At the end of this chapter we show a complete REA-DSL model according to our accompanying *Sy's Fish* example.

5.1 REA Notation Elements

In the following Figures all the notation elements used in the REA-DSL together with their descriptions are depicted. This section gives you a short overview of all the stencils used in the REA-DSL. *Stacked stencils* usually mean, that instead of exactly one occurrence there are one-to-many occurrences of the element. *Dashed stencils* usually indicate, that the REA concept is used on the type level.

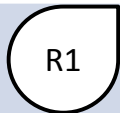

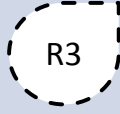


Notation	Name	Description
	Resource	Goods, rights, or services exchanged between agents in economic events.
	Resource Series	Indicating multiple resources of the same kind.
	Resource Type or Bulk Resource	Resources, where each individual resource cannot be tracked or where it is not economically traceable.
	Resource Type Series	Indicating multiple resource types or bulk resources.
	Labor Resource	Special service resource, where labor acquired by payroll processes is recorded. Consumed by events via participation by agents.

Figure 5.3: Resources Notation Elements

Figure 5.3 depicts resource stencils. All the resource stencils are depicted by a drop shape. Solid drops refer to identifiable resources and dashed drops refer to none-identifiable resources. In general, a stack of drops refers to multiple resources. This is especially important once the resources get connected to events they are used in. It defines, if multiple resources or just a single resource can be used in an event. Since labor is a special kind of resource in REA, it also becomes an extra stencil. This stencil is a dashed drop annotated by three small stick figures.









Notation	Name	Description
	Inside Agent	A person, company, or organizational unit inside a company.
	Outside Agent	A person, company, or organizational unit outside a company.
	Inside Agent Series	Multiple inside agents.
	Outside Agent Series	Multiple outside agents.
	Inside Agent Type	Type of an inside agent.
	Outside Agent Type	Type of an outside agent.
	Inside Agent Type Series	Multiple inside agent types.
	Outside Agent Type Series	Multiple outside agent types.

Figure 5.4: Agents Notation Elements

Figure 5.4 depicts agent stencils. Our REA-DSL consists of eight different agent stencils. They are actually combinations of three different kind of stencils: a white or black-head stick figure, a dashed or solid stick figure, and a single or stack of stick figures. White-head stick figures refer to agents inside our company, whereas the black-head stick figures refer to agents

outside of our company. A solid stick figure represents one specific agent (e.g., the specific salesman JOHN), whereas a dashed stick figure represents a type of an agent (e.g., anybody who is of the type salesman). This becomes especially important, once you want to reserve either a specific agent for a future event or if you do not care for a specific agent and you only reserve a yet-not-known agent who will be instantiated at the time of the event happening. If there is a stack of agents, it means that multiple agents are involved. This is important when agents participate in events. A stack of agents refers to multiple agents participating in the event and a single agent refers to a single agent participating in this event.


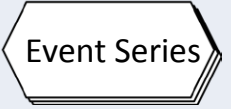


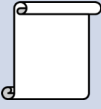
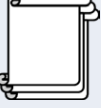
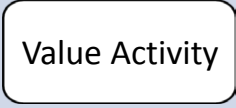
Notation	Name	Description
	Event	Events enable agents to exchange resources in economic exchanges.
	Event Series	Multiple events.
	Event Type	Type of an event.
	Event Type Series	Multiple event types.
	Commitment	An obligation to perform an event in the near future.
	Commitment Series	Multiple commitments.
	Value Activity	Either a transfer or a transformation of resources, that increases the value for the company.

Figure 5.5: Event/Commitment/Value Activity Notation Elements

Figure 5.5 depicts event, commitment, and value activity stencils. Events can be represented by four different kind of stencils which consist of a combination of solid or dashed hexagons, and a single or stack of hexagons. A solid hexagon represents the actual event happening, a dashed hexagon refers to a type of an event. A stack of hexagons refers to multiple events. This for example might specify that multiple sale events can be required by one overall payment event. A scroll and stack of scrolls refer to commitments made to execute events in the future. Value

activities, which correspond to REA business processes, are depicted by rounded rectangles and define an abstract transfer or transformation.

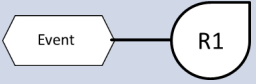
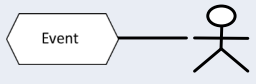
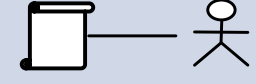
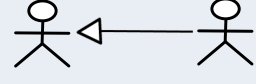
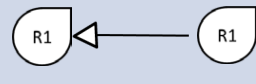
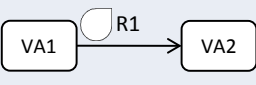
Notation	Name	Description
	Stock-Flow	Reserving a resource or specifying a (dashed) resource type or bulk resource.
	Participation	Reserving an agent or specifying an (dashed) agent type.
	Commit	Agent legally committing to a commitment to execute specific events in the future.
	Generalization Agents	Sub agents are specialized super agents. They inherit the attributes of the super agent.
	Generalization Resources	Sub resources are specialized super resources. They inherit the attributes of the super resource.
	Resource Flow	A resource flowing from the output of a value activity to the input of another value activity.

Figure 5.6: Association Notation Elements

Figure 5.6 depicts associations between the stencils of the REA concepts explained previously. There are six different kinds of relationships. A stock-flow relationship, a participation relationship, a commit relationship, a generalization relationship for agents as well as for resources, and a resource flow relationship. Stock-flows are depicted by a solid line and specify which resources (or multiple resources in a resource series) are exchanged in what events. Participate relationships are also depicted by a solid line and specify which agents (or multiple agents in an agent series) are participating in which events. A commit relationship is also represented as a solid line between a stick figure and a scroll and specifies which agent is committing to a future event. Generalization hierarchies of agents and resources are depicted by a white-head arrow between them (known from class diagrams). Lastly, resource flows are depicted by a black solid arrow annotated by a resource symbol pointing from one value activity to another. This indicates, that the resource is created by the source value activity and is consumed/used by the receiving value activity. We note that participate and stock-flow relationships are similar to the reserve and specify relationships explained later. The only addition is that they can also appear between types of events, agents, and resources. A more detailed description of all the elements introduced above is given in the following sections.

5.2 Agent View

In the REA ontology agents have control over resources and are able to transfer or transform them in an event. Additionally, they can be the stakeholders in a commitment, where agents bind themselves to execute certain events in the near future. Therefore, agents can also be reserved or specified for these future events. REA distinguishes between inside agents and outside agents. Inside agents are within the company and outside agents are outside of the company. In our REA-DSL the agents are defined in a specific view called the agent view. The agents specified in this view can be referenced by two of the other REA views, namely the operational view and the planning view explained later. In the following, the agent view is explained in detail by describing the underlying agent view meta-model (cf. Figure 5.7) and the concrete syntax of the agent view shown in an abstract example in Figure 5.8. The numbers in the circle help to match the element in the meta-model with the stencil in the concrete syntax.

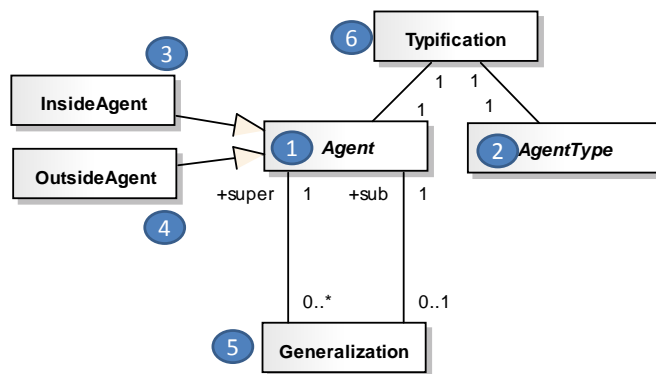


Figure 5.7: Agent Meta-Model

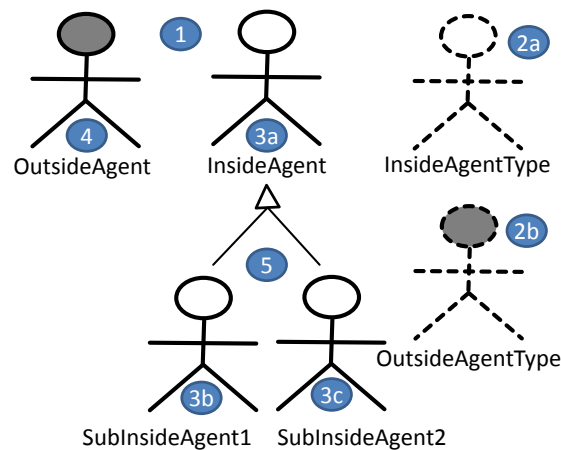


Figure 5.8: Agent Abstract Model

Agents and Agent Types. The main object in the agent view is an *agent* (cf. Figure 5.7 (1)). It is represented by a *stick figure* as shown in Figure 5.8. The meta-model specifies, that one agent (1) has exactly one *typification* (6) relationship to exactly one *agent type* (2) and vice versa. In the end this means that there is a one-to-one relationship between the meta classes *agent* and *agent type*. This one-to-one relationship is best explained when looking at an example from another area: in the object-oriented paradigm classes may comprise of static attributes (class attributes) and non-static (regular) attributes. In the REA context it is required to have one concept covering the regular attributes and another one covering the static attributes. For example `shop assistant` is an *agent* having regular attributes, whereas `shop assistant type` is an *agent type* covering all static attributes that are common to all `shop assistants`. Note, in this chapter we do not care about attributes, but they become vital in the relational mapping for accounting systems in the Chapter 6 "REA-DB".

Because of the one-to-one relationship between agents and types on M2, in the agent view a stick figure (1) represents the agent as well as implicitly its type, which does not need to be modeled separately. However, in the concrete syntax of the planning view which is introduced later on, agent types can be associated and are depicted as *dashed stick figures* (2). The planning view specifies if an event type relates to a type of an agent or to a specific agent. For example, in one case we just want to reserve an arbitrary salesperson and therefore use the agent type (2), but we do not care if it will be salesperson `Lisa` or salesperson `John` at the moment of reservation. In another case, we want to reserve a specific agent and thus, use a specific agent (1) (e.g., `Lisa`).

Inside and Outside Agents. According to the meta-model an *agent* (1) can either be an *inside agent* (3) or an *outside agent* (4). *Inside agents* are depicted by a *white-head stick figure* and are acting within the company. *Outside agents* are depicted by a *black-head stick figure* and are trading partners outside the company. As for *agent types* (2), the *white-head dashed stick figure* refers to an *inside agent type* and the *black-head dashed stick figure* to an *outside agent type*.

Generalization. *Agents* can form a *generalization hierarchy* (5) between the *sub agent* (3b, 3c) and the *super agent* (3a). The *generalization* defines an "is-a" relationship. A more general *agent* can be substituted by a more specific *agent* in a REA model (cf. [LW94]). The generalization is depicted in the abstract model by a *white-head arrow* (5) pointing from the *sub inside agent 1* (3b) to the *super inside agent* (3a).

5.3 Resource View

Ijiri [Iji75] defines resources as objects that are scarce and have utility and that are under the control of an enterprise. Resources can either be goods, rights, or services. REA distinguishes between resources that can be uniquely identifiable and resources that are not uniquely identifiable. Resources might be decremented or incremented in economic events, where one agent loses the control of this resource and another agent gains control of this resource. In our REA-DSL all the resources are defined in the resource view. The resources defined in the resource view can be referenced by the value chain view, operational view, and planning view explained later on. In the following, the resource view is explained in detail by describing the underlying

resource view meta-model (cf. Figure 5.9) and the concrete syntax of the resource view shown in an abstract example in Figure 5.10. Again, the numbers in the circle help to match the element in the meta-model with the stencil in the concrete syntax.

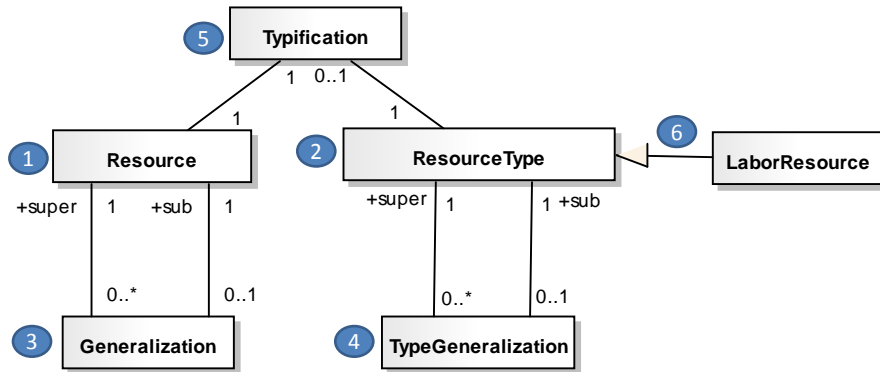


Figure 5.9: Resource Meta-Model

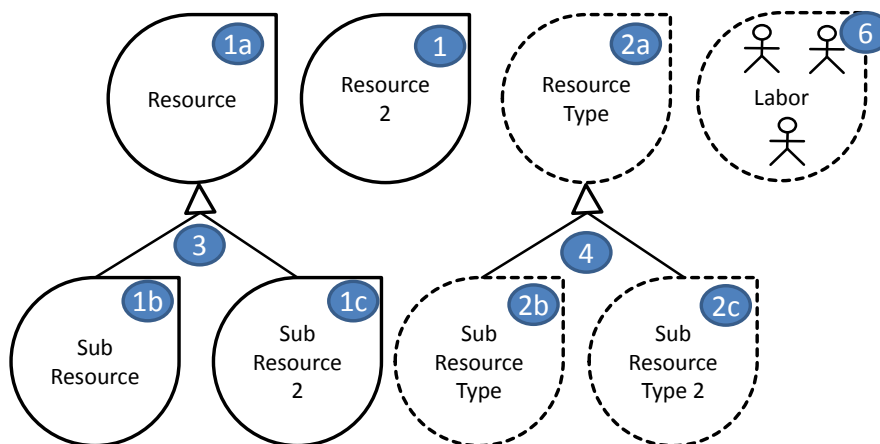


Figure 5.10: Resource Abstract Model

Resources and Resource Types. A *resource* (cf. meta-model in Figure 5.9 (1)) is depicted by the shape of a *solid drop* shown in the resource model in Figure 5.10. The *typification* (5) mechanism for *resources* is quite similar to the one of agents. The meta-model specifies, that one resource (1) has exactly one *typification* (5) relationship to exactly one *resource type* (2). However, a resource type can have, but does not have to have, a typification relationship with a resource. This will be explained in more detail in the next paragraph about *bulk resources*. Usually there is a one-to-one relationship between the meta classes *resource* and *resource type*. Thus, there is one concept covering the regular attributes and another one covering the static attributes. For example `product` is a *resource* having regular attributes where the value changes for each single instance (e.g., RFID number), whereas `product type` is a *resource type* cov-

ering all static attributes that are common to all `products` of a type (e.g., tax percentage). Because a resource always has a resource type, the resource type is always implicitly modeled together with the drop of the resource and thus, does not have to be modeled separately. However, a *resource type* might also be specified without a related *resource* indicated by the "0..1" cardinality which is discussed next.

Identifiable and Bulk Resources. *Resources* can be seen in two different ways: resources as an *identifiable resource* and resources as a *bulk resource*. *Identifiable resources* are resources, which can be identified and tracked individually (e.g., cars). Consequently, it is important to record each single occurrence of this resource in an information system. Thus, *identifiable resources* are modeled as regular *resources* (1) and are depicted by a *solid drop* in the concrete syntax. On the other hand, *bulk resources* are not tracked individually. These can either be resources which are not important to track individually or resources which cannot be tracked individually (e.g., water, nails). On closer examination, these *bulk resources* are nothing else than *resource types* (2), which group actual phenomena having common properties together. Thus, these *bulk resources* are modeled as *resource types* (2) and are depicted by a *dashed drop* in the concrete syntax. Consequently, they do not belong to a *resource* (1) as indicated by the "0..1" cardinality. *Bulk resources* must contain information about the quantity on hand and the measurement unit in order to keep track of the sum of resources, which are not individually tracked. This additional information is introduced in Chapter 6 "REA-DB".

Labor Resource. There is one special kind of a *bulk resource* called *labor resource* (6). The *labor resource* is often recorded in minutes for each inside agent. It is the service *resource* agents commit to provide in return for getting payed in the *payroll process*. In other words, if a company pays the employees, they get in return the resource labor. However, this *labor resource* is consumed/used in a different way than regular resources. Whereas regular resources are consumed/used when in a stock-flow relationship with a decrement event, the *labor resource* of a specific *inside agent* is consumed when the *inside agent* participates in *events*. *Labor* is depicted by a *dashed drop* with three little *inside stick figures* (6). There can only exist one *labor resource* in the resource view. This single *labor resource* contains the labor records for each inside agent.

Generalization. Similar to the agent *generalization hierarchy*, a *generalization hierarchy* might also be added to resource. *Generalization hierarchies* (3,4) can be assigned between *resources* (1) or between *resource types* (2). As in the meta-model, a generalization (3) is between a *super resource* (1a) and a *sub resource* (1b). The super resource is a more general resource than the more specific sub resource. Thus, it represents an "is-a" relationship. Each instance of the more specific resource is also an instance of the more general resource. Therefore, the more general resource can be substituted by the more specific one (cf. [LW94]). Accordingly, the meaning in REA is that a more specific resource might serve as an input for a more general resource for an event. For example, an event requires the *Resource* (1a), but may also receive the *Sub Resource* (1b or 1c). This same concept applies to the resource type of a resource: a more specific resource type might serve as a substitute for a more general resource type in an event. In the concrete abstract model the generalization for resources is depicted as a *white arrow* (3) pointing from the sub resource (1b) to the super resource (1a). A generalization (4) can appear between resource types (bulk resources) as well.

Adding Sub Resources During Runtime. One may not want to model all sub resources at design time or the sub resources may be dynamic and change during time. For example, we have the resource product modeled in the resource view without any sub resources. During the runtime of the information system based on this REA-DSL, it will still be possible to add new sub resources. For example, sub resources like `aquarium` or `fish knife` might be added having the resource `product` as super resource. According to the REA ontology, the new resource `aquarium` is of the new resource type `aquarium` and the new resource `fish knife` is of the new resource type `fish knife`. If in a REA model the resource product is referenced, it can also be substituted during runtime by the more specific resource `aquarium` or resource `fish knife`. Accordingly, this also applies to resource types (e.g., the product type being substituted by the `aquarium` type).

The concept of adding sub resources during runtime was not discussed for adding sub agents to the agent view during runtime. We argue, that the hierarchy of agents is stable throughout a long period of time and dynamically adding sub agents to an information system is not as relevant. However, it is supported by the REA-DSL.

5.4 Operational View

The operational view introduced in this section specifies which events "are happening" or which events "have happened". The concepts used for this view are taken from the REA operational layer. In general, it defines which agents exchange what resource in the course of what event. Furthermore, it might also specify transformations, where resources get transformed into other resources by the help of inside agents. An operational view consists of multiple duality models describing exactly one economic exchange with the exchanged resources and the participating agents. In the following, we describe the concepts of the operational view of our proposed REA-DSL.

Duality. The duality relationship is a core concept in REA. It links an increment in the resource set with a corresponding decrement; where increments and decrements must be members of two different event entity sets. In the REA ontology, the duality relationship is characterized by the unary relationship assigned to the concept of an economic event (see Figure 4.2).

In the REA-DSL, *duality* becomes a core model element that serves as a building block for further purposes. The operational view meta-model is depicted in Figure 5.11. Figure 5.12 presents a (rather abstract) example model - which helps to understand the meta-model concepts and to introduce the concrete syntax and the corresponding stencils. Concepts depicted in the abstract example are marked by a circled number identifying the corresponding class in the meta-model.

Transfer, Transformations, and Entity Sets. The *duality* (1) concept can either be a *transfers* (2) (exchanges with external actors) or a *transformations* (value creation inside the enterprise). No matter which kind of specialization it is, a *duality* always covers two distinct *entity sets*; the *increment set* (3) describing the increments in the resource sets and the *decrement set* (4) describing the decrements in the resource sets. Figure 5.12 shows an abstract example of a *duality* model to illustrate the concepts and their stencils. An entity set is modeled as a swimlane. Accordingly, a duality model includes two swimlanes. The *duality* (1) shown in the example is

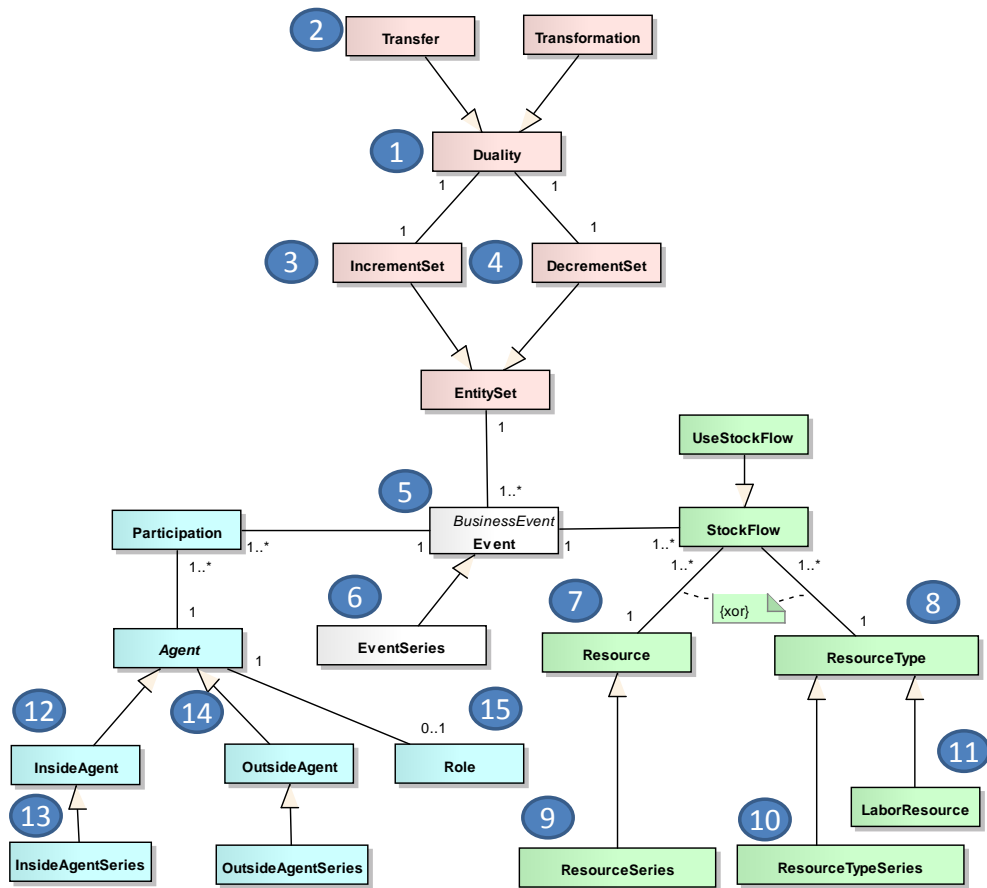


Figure 5.11: Operational View Meta-Model

a *transfer* (2). It follows that in the *decrement set* (4) we *give* or *use* resources and in the *increment set* (3) we *take* resources. The kind of *entity set* is denoted in the upper left corner of the swimlane.

Events. According to the meta-model, an *entity set* covers at least one but up to multiple *events* (5). An *economic event* is considered as a class of phenomena reflecting changes in scarce means [Yu76]. An *event* is specific to the *entity set* it belongs to. Following the principles of *duality*, all *events* in the *decrement entity set* are counterbalanced by the *events* in the corresponding *increment entity set* of the same *duality* model.

An *event* is usually executed at a certain point in time. However, in certain cases the increment/decrement is realized by a series of events each of the same nature but performed at different points in time. For example, consider that the payment for goods is split into a number of partial payments. For this purpose we use the concept of an *event series* (6), which is defined as a specialization of the *event* in the meta-model. Consequently, the concept of an *event series* may substitute for an *event*. This means, instead of modeling each *event* of each partial payment, one may model a single *event series* of partial payments.

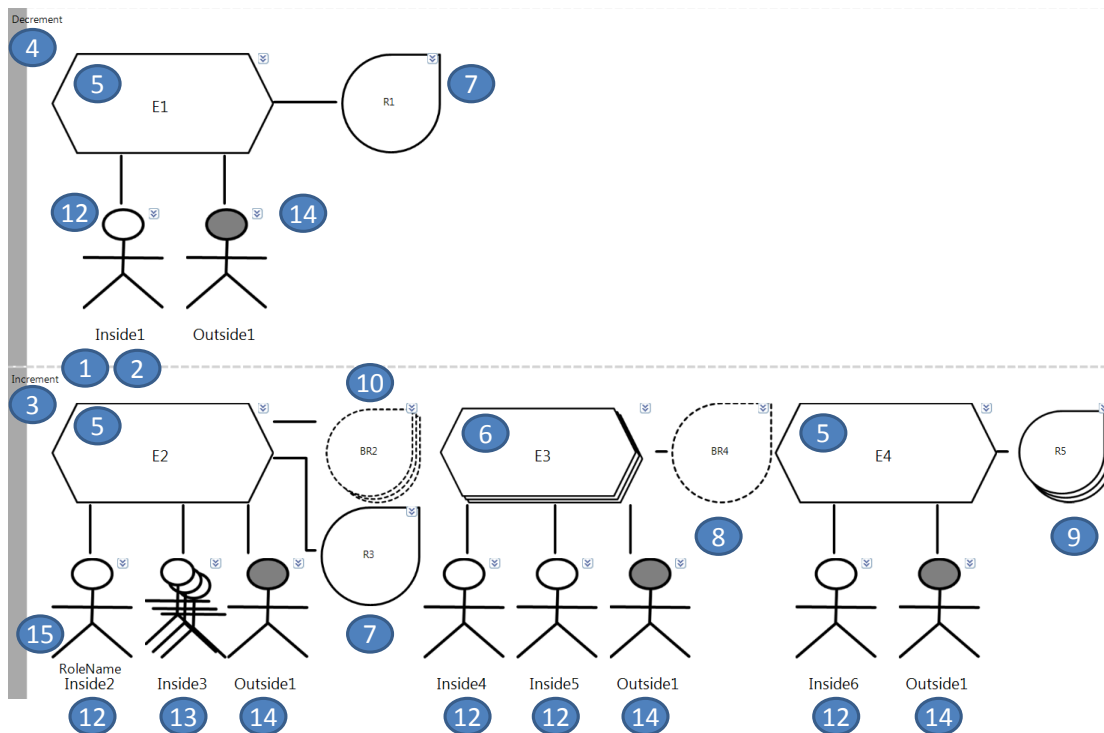


Figure 5.12: Operational View Abstract Example

Resources and Resource Types. An event in a decrement *entity set* decrements *resources (types)* (7,8). Similarly, an event in an increment *entity set* increments *resources (types)*. A resource (7) references an *identifiable resource* in the *resource view* and a *resource type* (8) a *bulk resource* in the *resource view*. The relationship between an event and a resource (type) is described by the concept of a *stock-flow*. A *stock-flow* models an association between exactly one event and exactly one resource (type). An event will usually affect one resource (type) only, but it may affect multiple ones. Thus, an event may have one up to many *stock-flows* connected. A resource (type) usually is affected by many different events (in different *entity sets* of different *duality models*). At a minimum a resource (type) is affected by one event - otherwise it would not be worth considering the resource (type) at all. Consequently, a resource (type) is connected to one up to many *stock-flows*. If a resource (type) is not consumed in an event but rather just used in an event, then this resource (type) is connected to the event by a *use stock-flow* relationship. *Use stock-flows* and regular *stock-flows* can be used together in one event (e.g., producing a pizza consumes the resources ingredients but uses the resource oven).

Resource Series and Resource Type Series. A similar concept to *event series* is defined for *resources (types)*: the *resource (type) series* (9,10). A *resource (type) series* is used if an event affects a number of *resources (types)* of the same kind. For example, on a high level of abstraction one may define raw material as a *resource (type)* and an event may affect a number of *resources (types)*. It is important to note that a resource (type) series may substitute a resource

(type) in a stock-flow, but the resource (type) series must not be used in resource (type) generalization hierarchies. A *resource (type) series* must always be based on exactly one existing *resource (type)*. For a *resource (type)* one may define zero to many *resource (type) series* (used in different *stock-flows*).

Resource Labor. As mentioned previously, *resource labor* (11) is a special kind of a *resource type* (8). It is the output of a *payroll value activity*. Consequently, in a *duality*, it can only appear on an *event* in the *increment set* (3). In Figure 5.13 an example of such a *duality* is shown. Since the *payroll value activity* looks quite similar in different business cases, we take a concrete example to explain the *labor resource* used in an *event*. In the *decrement event* cash disbursement (5a) the *bulk resource* cash (8) is paid to the *agent* employee (12b) from the *agent* cashier (12a). In return, the two *agents* clerk (12c) and employee (12d) participate in the *event* labor acquisition (5) where the *labor resource* (11) is acquired. This *labor resource* is decreased, when *inside agents* participate in an *event* explained in the upcoming agent paragraphs. Consequently, even during the labor acquisition *event*, *labor* is actually consumed by the participation of the *agents* cashier, employee, and clerk.

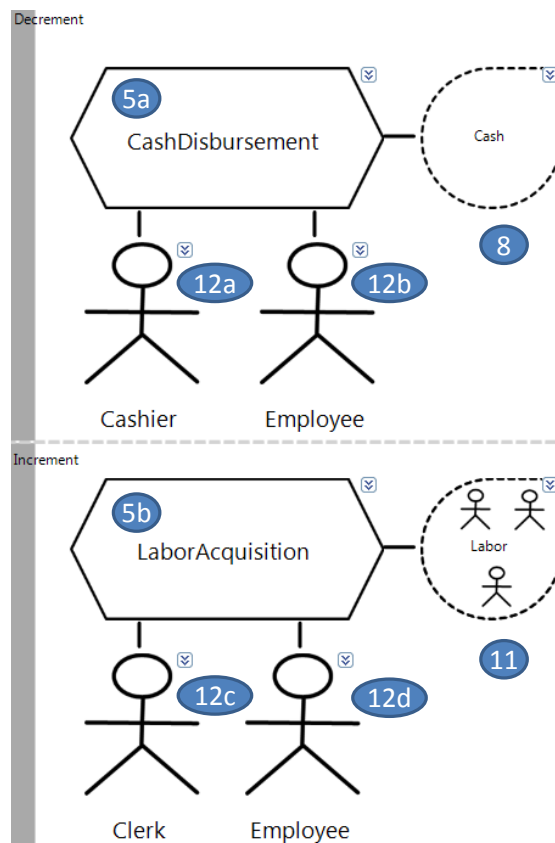


Figure 5.13: Labor Acquisition with Labor Resource

Inside Agents and Outside Agents. An *event* involves participating *agents*. We distinguish

between *outside agents* (14), i.e., trading partners outside the company, and *inside agents* (12) who are accountable inside the company. The involvement of *agents* in *events* is denoted by the concept of *participation*. A *participation* is an association that connects exactly one *event* with one *agent*. An event is associated to at least one, but up to many *agents*. Hence, an *event* has one to many participation associations. An *agent* participates in at least one, but up to many *events* (in the same, but also in different *entity sets* of the same or different *duality* models). Thus, an *agent* has one to many participations connected. If a participating *agent* is an *inside agent*, the time used in an *event* is subtracted from the *labor resource*.

Inside Agent Series and Outside Agent Series. A similar concept to *resource series* is defined for *agents*: the *agent series* (13) (depicted as a stack of stick figures) for *inside agents* and *outside agents*. An *agent series* is used if an *event* affects a number of *agents* of the same kind. For example, on a high level of abstraction one may define *shop assistant* as an *agent* and an *event* may affect a number of *agents*. It is important to note that an agent series may substitute an agent in a participation relationship, but the agent series must not be used in agent generalization hierarchies. An *agent series* must always be based on exactly one existing *agent*. For an *agent* one may define zero to many *agent series* (used in different *participation* relationships).

An agent might as well define a *role* (15). This helps to differentiate between same *agents* (*types*) connected to a single event. Consider two agents *guide* connected to one event *expedition*. We can give the one *guide* the role *leader* and the other *guide* the role *assistant*. This way, we can give each agent *guide* a specific meaning. Evidently, we can also apply roles to an *agent series*. In the case of an *agent series* for the *guide* with the role *assistant*, we might have multiple *assistant guides*.

In addition, there are further constraints assigned to the meta-model by OCL constraints to handle specifics of *transfers* and *transformation*. These OCL constraints are specified in Appendix B.2. In case of a *transfer*, each *event* must be assigned to exactly one *outside agent* and, in addition, to at least one *inside agent*. In general, *events* of the same *transfer* (both in the *decrement* and the *increment entity set*) must involve one and the same *outside agent*. However, there are some rare cases where different *agents* and *agent series* might be used (e.g., outside agents representing different organizational units of a company). In the case of a *transformations*, there are no outside agents allowed to participate in the event.

Operational View Abstract Example. The relationships among *events*, *resources*, and *agents* within an *entity set* is also illustrated in the abstract example of Figure 5.12. *Events* are denoted by an hexagon (5). In the *decrement/give* swimlane (4), there is only one *event* E1. This *event* leads to a decrement of the only associated *resource* R1, notated by a drop (7). Since the *duality* model (1) is a *transfer* (2), exactly one *outside agent* Outside 1 and an *inside agent* Inside 1 is involved. The symbol for *agents* is a stick figure (12,14) - *outside agents* have a black head (14), whereas *inside agents* have a white head (12).

The *increment/take* swimlane (3) includes three *events* E2, E3, and E4; one of which (E3) is an *event series*. The symbol of an *event series* is a stack of hexagons (6). All three *events* together compensate the single *event* E1 of the *decrement/give* swimlane. E2 is associated with the *bulk resource series* BR2 (depicted by a stack of dashed drops) (10) and the resource R3 that are incremented, whereas the series E3 increments only the *bulk resource* BR4 (depicted by a dashed drop) (8). E4 leads to an increase of the *resource series* R5. A *resource series* is depicted

by a pack of drops (9). Given the example of a *transfer*, all three events E2, E3, E4 in the *increment/take* swimlane are associated with the same *outside agent* Outside 1 (14) as the event E1 of the *decrement/give* swimlane. In addition, E2 involves one *inside agent* Inside 2 with a *role* Role Name (15) placed above the agent name, and multiple *inside agents* Inside 3 (depicted by the dashed stick figures) (13). Lastly, E3 involves two different *inside agents* Inside 4 and Inside 5 and E4 only one *inside agent* Inside 6.

5.5 Value Chain View

According to Geerts and McCarty [GM97], the duality relationships introduced previously are the glue that binds a company's economic events together into rational economic processes, while "stock-flow" relationships weave these processes together into an enterprise value chain. In the latter case, they do not refer to the stock-flows within a single duality model. Rather, they mean that an increment in a resource as a result of one duality model (i.e., business process) will serve as a chance to decrement this resource in a subsequent duality model. In other words, a value chain is built by a well defined series of duality models. The transitions between the duality models reflect the resource dependencies between the duality models.

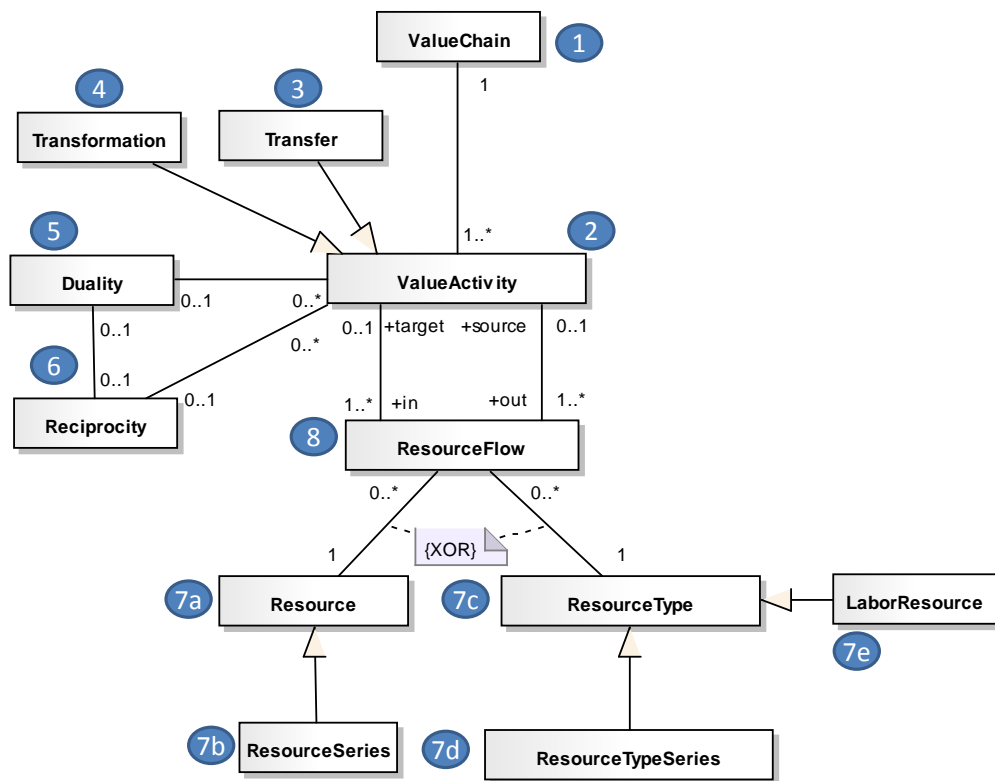


Figure 5.14: Value Chain View Meta-Model

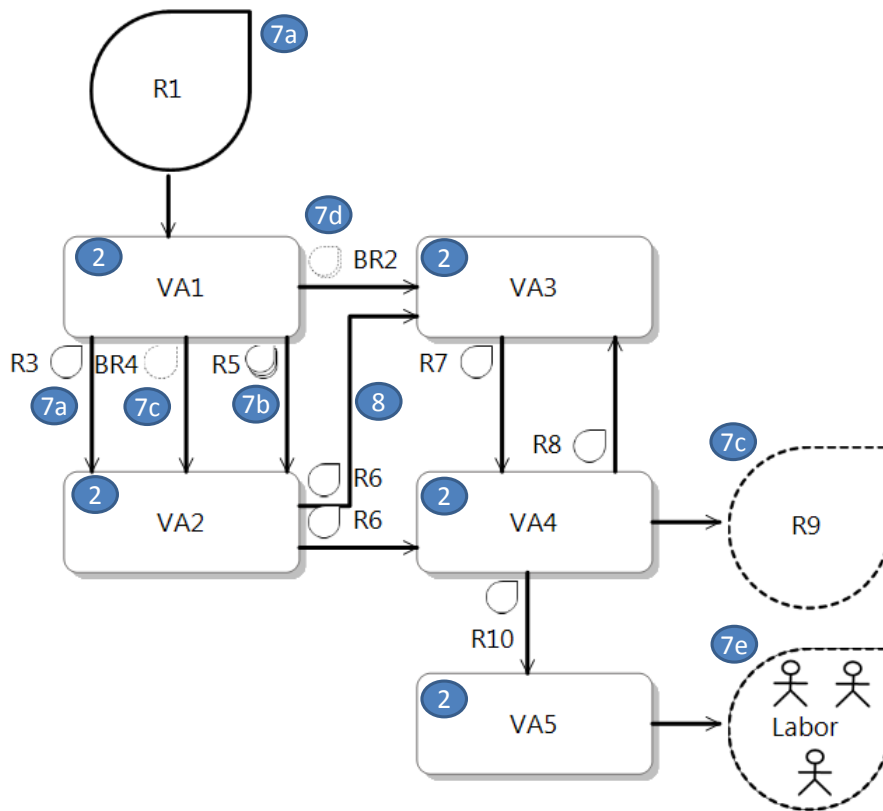


Figure 5.15: Value Chain View Abstract Example

Value Activities. These general ideas are reflected in the value chain meta-model of Figure 5.14. A *value chain* (1) is built by a number of *value activities* (2). A *value activity* is used only once in one distinctive *value chain* and is defined as a *transfer* (3) or a *transformation* (4). It follows a *value activity* may point to an underlying *transfer/transformation* described by a *duality* (5) model in the operational view and/or to an underlying *contract/schedule* described by a *reciprocity* (6) model in the planning view, which is introduced in the next section. The underlying *duality* model specifies the *value activities* in more details including the *agents* participating. If commitments are defined for this duality model, the *value activity* also references a *reciprocity* model capturing these commitments. A *duality/reciprocity* model is usually the basis of one value activity, but may be referred to by multiple *value activities*.

Resources. *Resources* (7a), resource types (7c), resource series (7b), resource type series (7d), and labor resources (7e) are depicted by a drop and tie the *value activities* together. Thus, an *resource flow* (8) - which points to exactly one *resource (type)* - connects two *value activities*. A *resource flow* is a directed association that usually starts from a source *value activity* and ends at a target *value activity*. However, we also allow for *resource flows* that have either no source *value activity* or no target *value activity*. This allows for a partial analysis, when one considers a certain *resource (type)* as given or when an *resource (type)* is considered as final output of the

value chain. Typically, cash is often assigned to such *resource flows*. A resource type (depicted by a dashed drop) actually refers to a *bulk resource* defined in the resource view. *Resource series* and *resource type series* – either depicted as a stack of solid drops or stack of dashed drops – refer to many *resources* or many *resource types*, respectively.

Labor Resource. The *labor resource* ($7e$) is a special kind of a *bulk resource* recording the time of available labor by the *agents*. It has to refer the *labor resource* defined in the *resource view*. Usually, the *labor resource* serves as an input *resource* for each single *value activity*. However, it is easy to imagine how overloaded the value chain view would become, if we would depict the labor input for each single value activity. Consequently, we define, that each value activity implicitly contains the labor resource as an input but is not depicted in the value chain view explicitly. Evidently, the *labor resource* might only be defined as the output of a *value activity* (i.e., the payroll process). The *labor resource* is consumed by the underlying *dualities* of the *operational view* when *agents* participate in *events*.

Resource Flows. A *value activity* has at least one, but up to many outgoing *resource flows*. Similarly, a *value activity* has at least one, but up to many incoming *resource flows*. In order to deliver a consistent *value chain* two important constraints on the value activities have to be considered: For each *resource (type)* in the *decrement entity set* of the underlying *duality/reciprocity* model, at least one corresponding incoming *resource flow* pointing to the same *resource (type)* must exist. In analogy, for each *resource (type)* in the *increment entity set* of the underlying *duality/reciprocity* model, at least one corresponding outgoing *resource flow* pointing to the same *resource (type)* must exist. However, one may consider the substitutability concept of more general and more specific *resources (types)* as defined in a resource generalization hierarchy. In other words, a *duality/reciprocity* model expecting an *resource (type)* in the *decrement entity set* may also accept a more specific *resource (type)* in the incoming *resource flow* and an *increment entity set* may accept a more specific *resource (type)* on the outgoing *resource flow*. In a special case a source *value activity* of a *resource flow* outputs a *resource (type) series*, whereas the target *value activity* expects a single *resource (type)*. In that case, a *resource flow* might also carry a *resource (type) series* from a *source value activity* to the *target value activity*. Even though the *resource flow* defines a *resource (type) series*, the underlying *event (type)* may just refer a single *resource (types)*.

An example of the previous constellation is given in Figure 5.16. On the top the value chain depicts a car purchasing value activity and car a sale value activity. The car resource flow between the two value activities is depicted by a stack of drops indicating multiple cars. On the bottom of this figure, the incrementing event of the car buying value activity and the decrement event of the car selling value activity are shown. The car received event receives multiple cars, which is in accordance to the value chain. However, the car sale event just specifies one car, because we can just sell one car in a car sale event. Thus, the car sale event specifies a more specific constraint on the resource flow of the corresponding car selling value activity by just allowing a single car.

Value Chain Abstract Example. In Figure 5.15 we depict an abstract example of a *value chain* to illustrate its concepts and their stencils. Our *value chain* example includes five *value activities* VA1, VA2, VA3, VA4, and VA5 (2). The symbol for a *value activity* is a rounded rectangle (2). *Resource flows* are depicted by an arrow with the *resource (type)* assigned to this

arrow (8). For example, the *resource type* BR2 (a bulk resource depicted by a stack of dashed drops) flows from VA1 to VA3 (7d). However, for aesthetic reasons, we provide an alternative (but still similar) notation for *resource flows* that do not start from or do not end in a *value activity*. In this case, the resource flow arrow may start from the *resource (type)* (7a - big drop) or may lead to the *resource (type)* (7c - big drop), instead of assigning the *resource (type)* on the resource flow arrow. Examples are the resource flows of R1 leading into VA1 and of R9 starting from VA4.

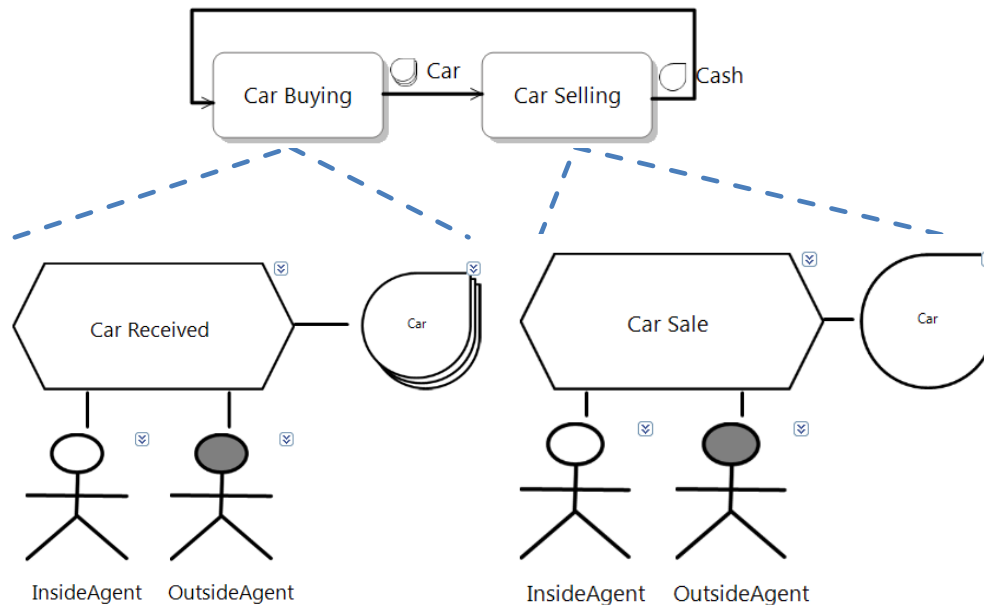


Figure 5.16: Value Chain Multiple Resource to Event Single Resource

VA1 points to the *duality* model of Figure 5.12. In this *duality* model the *resource* R1 sits in the *give-swimlane*; and the *resources* BR2 (*bulk resource series*), R3 (*resource*), BR4 (*bulk resource*), and R5 (*resources series*) are included in the *take-swimlane*. This is consistent with the *resource flows* to/from VA1 in the *value chain* model of Figure 5.15. VA1 receives R1 and delivers BR2, R3, BR4, and R5. It should be noted that the number of ingoing/outgoing transitions does not necessarily meet the number of *resources (types)* in the *duality* model, since a *value activity* may provide an *resource (type)* or (more unlikely in practice) receive a resource from multiple value activities. For example, VA2 provides R6 to both VA3 and VA4. The *value activity* VA5 leads to the *labor resource* (7e). Accordingly, in a concrete business example, VA5 is some kind of a payroll activity, which is further specified by a payroll duality like the one depicted in Figure 5.13. As mentioned above, the *labor resource* actually also serves as input to the *value activities* VA1, VA2, VA3, VA4, and even VA5, but is not explicitly modeled and depicted there.

Generalization Substitution. Let us assume that the *duality* model on which VA2 is based requires the *resources* BR4, R5, and R7. In this case one might think of an inconsistency since VA2 actually receives BR4 and R5, but substitutes R3 for R7. However, if R7 is defined as a

specialization of R3 in a resource generalization hierarchy, the model is consistent since in this case R7 may substitute R3 in the *duality* model of VA2.

Resource Series Refined by Single Resource. The *resource flow* between the *value activities* VA1 and VA3 defines a *bulk resource type series* BR2. This means, that there is one event in the underlying duality of VA1 receiving the *bulk resource type series* and one event in the underlying duality of VA3 consuming/using the *bulk resource type series*. However, the explained underlying events might also just refer to a single *bulk resource type* instead of a *bulk resource type series*. This way, it is possible to model a resource flow with a *bulk resource type series*, where the source *value activity* outputs multiple resources and the connected target *value activity* just consumes/uses a single bulk resource (or vice versa). For example, a production job value activity might create multiple products in the course of one event, but the selling value activity just sells one product in one selling event.

5.6 Planning View

5.6.1 Meta-Model and DSL

We have already described *exchanges* including *events* in the *operational view*. These exchanges often do not happen unexpectedly. There can be *commitments* to fulfill events in the future. Such commitments can for example be an `order to buy fish` or a `schedule to clean fish`. Thus, events happening in the future are planned beforehand. Therefore, we introduce an additional view called the *planning view*. This view makes use of the previously introduced views for modeling agents and resources. Similar to duality models in the operational view, the planning view consists of multiple reciprocity models. In the following section we propose the *planning view* meta-model (cf. Figure 5.17) and its concrete syntax (cf. Figure 5.18). Classes in the meta-model which also have a corresponding stencil in the concrete syntax of the abstract model are marked with a numbered circle.

Reciprocity. The root element of the planning view is the *reciprocity*. A reciprocity is the planned analog to a *duality* in an operational view. Duality connects the incrementing entities with the decrementing entities. Likewise, reciprocity connects the *incrementing commitments* with the *decrementing commitments* in the planning view. The reciprocity can be associated with a duality which refers to the corresponding duality in the operational view and vice versa. Thus, it is the link between the planning view which plans future events and the operational view which considers the events at the time of economic execution.

Reciprocity can either be a *contract* (1) or a *schedule*. A *contract* is made in advance to an upcoming *transfer*, and a *schedule* is made in advance to an upcoming *transformation*. The abstract example shows a contract (1) (indicated by `contract` in the brackets) with the name `contract name`. In the case of a schedule the term in the brackets would be `schedule`. If the reciprocity is a contract the corresponding duality in the operational view has to be a transfer and if it is a schedule the corresponding duality has to be a transformation, respectively.

Increment Plan and Decrement Plan. The reciprocity consists of two *plan entity sets* called *increment plan* (2) and *decrement plan* (3). The increment plan is depicted as the lower swimlane annotated by the term *increment* and contains all the commitments and related event

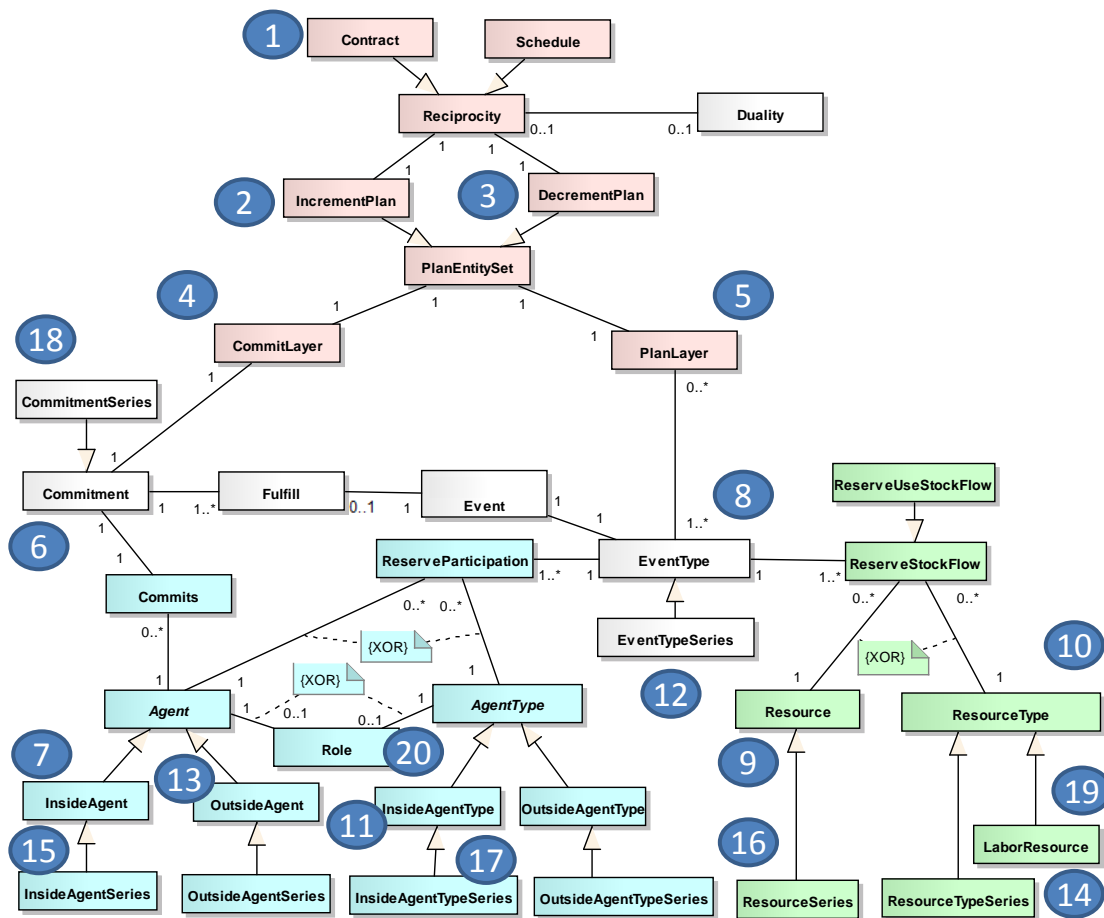


Figure 5.17: Planning View Meta-Model

types which will lead to an increment of resources in the future. Similarly, the decrement plan is depicted as a swimlane annotated by the term *decrement* and contains all the commitments and event types which will lead to a decrement of resources in the future.

The *increment plan* (2) and *decrement plan* (3) are divided into two layers: the *commit layer* (4) on the left side and the *plan layer* (5) on the right side.

Commit Layer. In general, the commit layer contains the *commitment* (6) which is made on future events defined by *event types* (8) in the plan layer. One *agent* legally *commits* to the *commitment* (6) which is depicted by a scroll. Agents can either be *inside agents* (7) or *outside agents* (13). *Inside agent series* and *outside agent series* are not allowed in commitments and are restricted by OCL constraints. The OCL constraints for the planning view are specified in the Appendix B.2. A commitment on the decrement plan always has to be in reciprocity with a commitment on the increment plan in order to provide the rational of individual economic activities. In a *contract*, an *inside agent* (7) has to commit to the commitment in the decrement plan and an *outside agent* to the commitment in the increment plan. As for a *schedule*, only

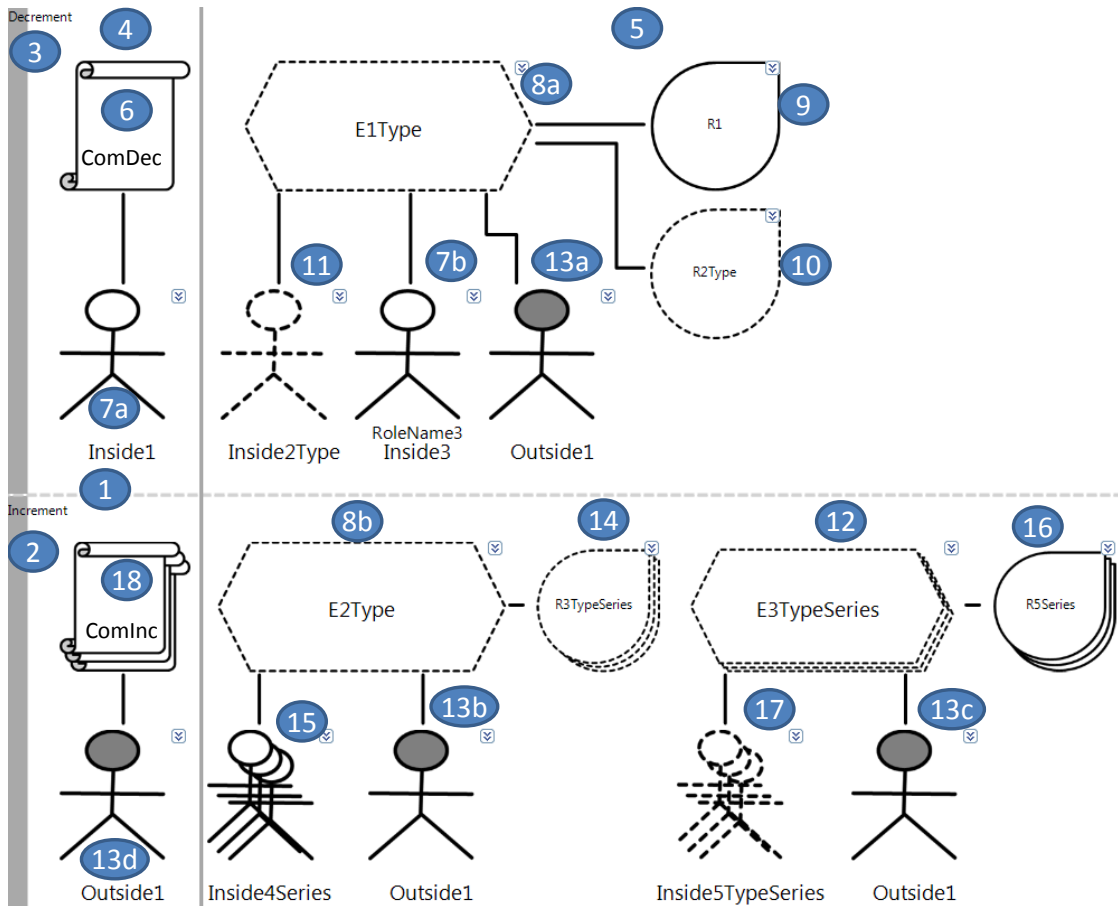


Figure 5.18: Planning View Abstract Example

inside agents can commit to the *commitment*. These restrictions are defined by OCL constraints as well. In the abstract example a commitment `comdec` (6) is made in the decrement plan (3) (upper lane) by the *inside agent* `inside 1` (7a) to execute *event types* (8) specified in the plan layer in the future. In return, a commitment `cominc` (18) is made by the *outside agent* `outside 1` (13d) in the increment plan (2) (lower lane) to execute *event types* (8) specified in the plan layer in the future. The commitment `cominc` (18) is actually a *commitment series* depicted by a stack of scrolls (18). The special meaning behind a commitment series is, that many commitments can be fulfilled by one event (e.g., two or more orders are fulfilled by a joint delivery).

Plan Layer. The *plan layer* (5) on the right side in general specifies the type of future *events* as well as their involved *agents* and *resources*. These future *events* are defined in the corresponding *duality* of the operational view.

Event Types. The plan layer contains one-to-many *event types* (8) depicted as dashed hexagons. Contrary to the *operational view* where *events* are identified, in the planning view only *event types* are specified (e.g., regular sale type). This is due to the fact, that at the time

of planning the events, the actual future events cannot be referred to, because they simply do not exist yet. Thus, only the event type can be referred to. Instead of event types, the sub type *event type series* (12) depicted by a dashed stack of hexagons might be specified in a plan layer. An event type series specifies one-to-many event types of the same kind (e.g., a payment being split up in many partial payments). The event type is related to the *event* in the duality which *fulfills* a commitment (6) or commitment series (18) in the future. A commitment can be fulfilled by one-to-many events, and one event fulfills zero-to-one commitments. This means, that for a commitment, there must be at least one related event in the operational view, but for an event, there does not necessarily exist a commitment. In the abstract example E1 type (8a) and E2 type (8b) are regular event types and E3 Type Series (12) is an event type series. Accordingly, an event E1 and E2 as well as an event series E3 Series have to exist in the operational view.

We note that an event type in the REA-DSL eventually becomes a commitment table in the relational table. This is due the fact, that the agent in the commitment actually commits to do the specific event type in the future. For example, in Figure 5.18 the agent *Outside1* (13d) commits to perform the event type *E2Type* and the event type *E3TypeSeries* in the future. Thus, there is actually one commitment to *E2Type* and one commitment to *E3TypeSeries* which is reflected in the relational tables accordingly. We describe this in more detail later in Chapter 6 "REA-DB". Thus, in the following, resources (types) and agents (types) connected to the *event types* are actually specified or reserved in a commitment.

Resources and Resource Types. *Event types* referenced by a *commitment* are connected to at least one *resource* (9) or *resource type* (10) by *reserve stock-flows*. These resources/resource types refer to resources/resource types in the resource view. If the event type resides on the increment plan (2) the resource will be gained in the future. Otherwise, on the decrement plan (3) the resource will be used in the future when connected by a *reserve use stock-flow* and consumed when connected by a regular *reserve stock-flow*.

A *resource* (9) connected to the event type specifies, that the exact identifiable resource is already known at the time of the commitment (e.g., a product with a specific RFID code). Therefore, the resource is *reserved* by the commitment. Consequently, the exact same resource is also referred to in the future event of the operational view. It is depicted by a regular *solid drop* (cf. 9, R1) and references an identifiable resource in the resource view. Similar, a *resource series* (16) has the same meaning for many resources of the same kind and is depicted by a *stack of solid drops* (cf. 16, R5 Series). It also references an identifiable resource in the resource view.

A *resource type* (10) connected to the event type is specified. This connection can have two specific meanings. Either the referenced resource in the *resource view* is an (i) *identifiable resource* or it is a (ii) *bulk resource*. In the case of an identifiable resource (i), in the future event of the duality a specific resource of this type will be associated. However, at the time of the commitment the exact identifiable resource is not-yet-known (e.g., booking a double bed room in a hotel, but the specific room with number 704 will be assigned at the time of arrival). This resource is depicted by a *dashed drop* (10, R2 Type). Similar, a *resource type series* (14) has the same meaning for many resource types of the same kind and is depicted by a *stack of dashed drops* (14, R3 Type Series). In the case of a referenced bulk resource (ii), the actual

resource can never be individually identified. Thus, it only defines the quantity of a resource type to be in a stock-flow of a future event (e.g., ordering 1000 liters of heating oil). Similar, a *resource type series* defines the quantity of many resource types to be in a stock-flow of a future event (e.g., ordering 1000 liters of heating oil and 70 liters of diesel).

Resource Labor. As mentioned previously, *resource labor* (19) is a special kind of a *resource type* (10). It is the output of a *payroll value activity*. Consequently, in a *reciprocity*, it can only appear on an *event type* in the *increment plan* (3). Usually the *labor resource* is received by a *labor acquisition* event type (cf. Figure 5.19). The structure of the complete reciprocity of this planning view is similar to the structure of the duality of the operational view depicted previously in Figure 5.13. The difference to the operational view is that in the planning view the *agent employee* commits to provide labor and in return the *agent clerk* commits to provide cash. This *labor resource* is planned to be used, when *inside agents (types)* are reserved/specified in a future *event type* explained in the upcoming agent paragraphs.



Figure 5.19: Labor Acquisition Event Type with Labor Resource

Agents and Agent Types. *Agents (series)* and *agent types (series)* are reserved/specified for participation in *event types* (8) through *reserve participation* links. *Agents* might either be *inside agents* (7) (white-head stick figure) or *outside agents* (13) (black-head stick figure). Reserved *agents* mean, that the individual agent can already be defined at the time of the commitment, and these are depicted by a *solid stick figure* (7b, Inside 3, 13, Outside 1). On the other hand, specified *agent types* mean, that only the type of agent can be defined at the time of the commitment, and these are depicted by a *dashed stick figure* (11, Inside 2 Type). Once the commitment is fulfilled by a future event, this *agent type* becomes a concrete known *agent*. An example is specifying that two arbitrary *shop assistants* are needed in the future event, but we do not know the exact individuals yet. An *agent series/agent type series* always refers to one-to-many *agent/agent types* and is depicted as a *stack of stick figures* (15, Inside 4 Series, 17, Inside 5 Type Series).

Event types of *schedules* have at least one *inside agent* (7)/*inside agent type* (11) and no *outside agents*. On the other hand, *event types* of *contracts* (1) additionally have *outside agents* (13). There is usually the same single *outside agent* used for every commitment and event type. However, in some rare cases there might be different *outside agents* as well as *outside agent types*. If a participating *agent* is an *inside agent*, the time he or she is planned to be used in an *event type* is scheduled. Once the actual event is happening, the actual used time will be subtracted from the *labor resource*.

The concept of *role* (20) is similar to the concept of role in the operational view. However, in the planning view, the role may be assigned not only to an *agent*, but also to an *agent type* or an *agent type series*. Consequently, this helps to differentiate between same *agents (types)*

connected to a single event. Applying roles on an agent type works in the same way as applying roles on the agent, which was already discussed for the operational view. Reserved *agents* will also be the concrete *agents* of the same *role* when the event happens; specified *agents* will become concrete *agents* of that *role* at the time of the *event*.

5.6.2 Mapping Between Planning View and Operational View.

The appearance of the planning view is quite similar to the concepts of the operational view. This is because of the relationship of the concepts in the planning view and the operational view. In the planning view, commitments are defined for types of events, which will happen in the future. These future events are exactly described in the operational view, and they are related to the event types specified in the corresponding planning view. Thus, operational layer stubs can be derived from the plan layer of the planning view and can be seen as a copy of the entity set with following adaptations: a series in the planning view is always mapped to a series in the operational view, and a single concept in the planning view is always mapped to the single concept in the operational view. An *event type* (8) becomes an *event*, an *event type series* (12) becomes an *event series*, a *resource* (9) stays a *resource*, a *resource series* (16) stays a *resource series*, an *agent* stays an *agent* and an *agent series* stays an *agent series*. As for the types, *agent type* and *agent type series* become *agent* and *agent series*, respectively. For a *resource type* (10) and a *resource type series* (18), it depends on whether the referenced *resource* in the resource view is an (i) *identifiable resource* or a (ii) *bulk resource*. In the case of an (i) *identifiable resource*, they become a *resource* or *resource series*. In the case of a (ii) *bulk resource*, they remain a *resource type* or *resource type series*. Accordingly, the event types and related agents/agent types, resources/resource types have to relate to the corresponding events, agents, and resources defined in the duality of the operational view. However, this is only a proposed mapping for generating a stub. Modelers might still change the operational view to suit their needs. For example, they can add an additional agent which was not important to record in the commitment.

If the planning view and the operational view do not differ at all in their concepts, they are conceptually congruent [RM99]. In that case, the operational view can be completely derived from the planning view without any modifications.

5.6.3 Policies Contained in the Planning View

Policies describe economic activities that should, could, or must happen in a company [GM06]. In our REA-DSL we are also able to describe basic concepts of the policy layer. The information for these policies are actually captured by our planning view. Thus, later on in Chapter 6 we are able to derive policies database structures from policies which are conceptually congruent to the planning view (having same relationships between concepts). We will assume that the planning layer describes that an order for the event selling a fish requires one salesman and one customer. A conceptual congruent policy for this planning view would put certain constraints on this order.

Examples for policies are shown in the rectangles in Figure 5.20. One example for such a policy might be: selling a fish in a specific region (e.g., Austrian sale event type, US sale event type) requires a specific salesman who speaks a specific language and therefore, qualifies

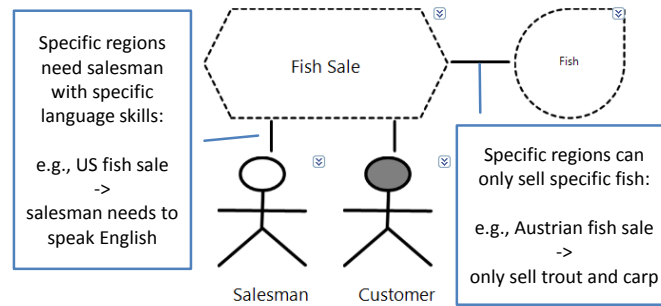


Figure 5.20: Policy Example

for this sale event type. Another policy can be, that only specific fish can be sold in specific regions. For example, only trout and carp are allowed to be sold in an Austrian sale and tuna are forbidden to be sold in an Austrian sale. This becomes more evident and clear once properties are introduced for planning view stock-flows and participate relationships in Chapter 6 "REA-DB". Thus, further examples concerning the policies can be found in the aforementioned chapter at Figure 6.4. Note, in the REA-DSL, only the structure of policies are defined, actual constraints are stored in the database during runtime.

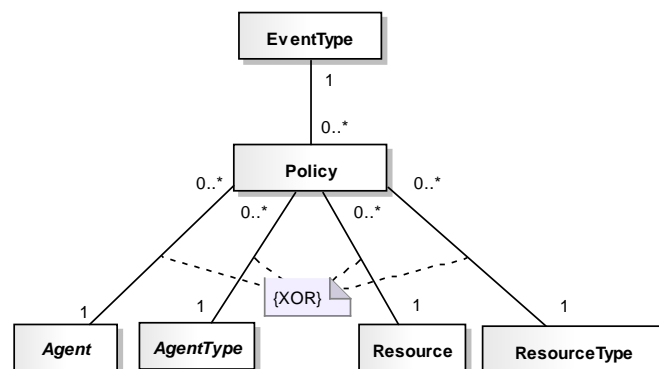


Figure 5.21: Policy Meta-Model

Figure 5.21 shows the basic meta-model for the policy layer. A more complex policy structure is out of focus for this thesis and would require a separate policy view. The meta-model describes, that a policy may exist between one event type and either one agent, agent type, resource, or resource type. Event types, agents, agent types, resources, and resource types may be part of multiple policies. To derive policies from the planning view, we describe the relations between the planning meta-model (cf. Figure 5.17) and the policy meta-model (cf. Figure 5.21) as follows: Event types, agents, agent types, resources, and resource types stay the same. A *series* gets reduced to the corresponding single concept of REA. The *reserve participation* and *reserve stock-flow* relationship both become *policy* relationships.

Looking at the event type `E1Type` in the abstract example of Figure 5.18, the following

policies might be derived: two policies between `E1Type` and the resources `R1` and `R2Type` as well as three policies between `E1Type` and the agents `Inside2Type`, `Inside3`, and `Outside1`.

5.7 Example

The accompanying example in this thesis is based on *Sy's Fish* - an example that was used also by Geerts et al. [GM97] to demonstrate REA. We introduced this example in Chapter 4, and we will give just a brief summary here. After that, we model this example according to the REA-DSL specified in this chapter.

Example Business Model. *Sy's Fish* purchases fish from the fish market. He sells three different kind of fish: carp, trout, and tuna. Additionally, he acquires products from a factory: books and fishing rods. The fish and products are transported to *Sy's Fish* by leased trucks. At *Sy's Fish* all the fish get cleaned. The products and fish are then sold to private customers and restaurants in order to make profit. To be able to accomplish all tasks, *Sy's Fish* employs a couple of employees, who can be a salesman or a shop assistant or a cashier.

Modeling REA. The REA-DSL consists of five different views: (i) *agent view*, (ii) *resource view*, (iii) *value chain view*, (iv) *planning view*, and (v) *operational view*. There is actually no specific order for modelers to create the views, and they are always able to switch between the different views. A modeler might create a resource the first time in the operational view and then add it to the resource view. However, we believe that it is favorable to first define all the building blocks of REA (i.e., the resources and agents in the resource and agent view). Then, to gain a good overview of the value activities the value chain should be modeled. After that, the value activities can be refined by the operational view for capturing current events. Since the value chain view already contains information for the operational view (i.e., the resources which are exchanged or transformed), a stub for the operational view can be created automatically. If we also want to model commitments in an additional step, the planning view for a value activity can be created. Similar to the operational view, a stub for the planning view can also be created from the value chain view. In the following, we will model these views one by one for *Sy's Fish*.

5.7.1 Agent View

We start modeling the business with the REA-DSL by defining the different agents. In the agent view (cf. Figure 5.22), the agents inside the company are depicted on the left side with the white-head stick figures. The general agent is an `employee`, which can either be a `salesman`, a `shop assistant`, or a `cashier`. These agent are modeled explicitly, because they accomplish different tasks for the company. The outside agents are depicted as black-head stick figures. *Sy's Fish* does business with the general agent `customer` which can either be a `restaurant` or a `private customer`. Furthermore, we interact with the outside agent `truck rental` for renting our trucks, the outside agent `factory` for purchasing products, and the outside agent `fish retailer` for buying fish.

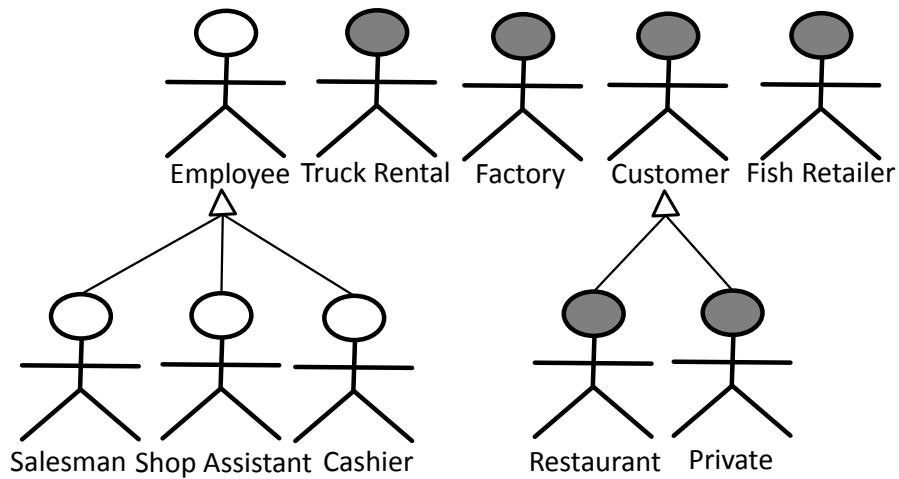


Figure 5.22: Sy's Fish Agent View

5.7.2 Resource View

Next, we define all resources which need to be tracked and recorded in *Sy's Fish* company. These resources are depicted by the shape of a drop in Figure 5.23.

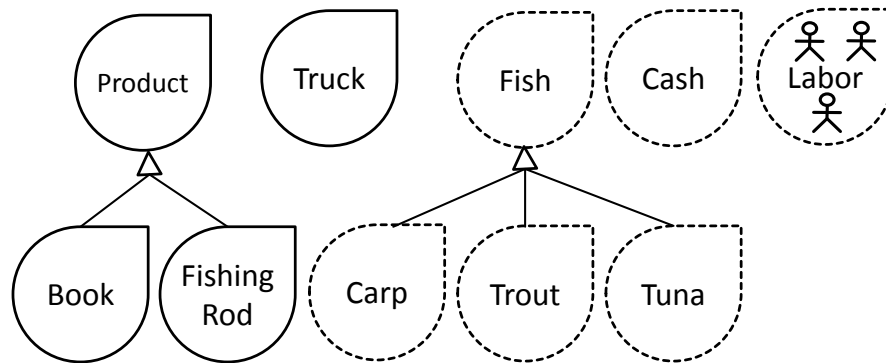


Figure 5.23: Sy's Fish Resource View

Resources which can be identified individually are marked by a solid drop, whereas bulk resources where the individual real-world object cannot be identified are marked by dashed drops. Accordingly, the resource `product` is a solid drop. Products can be categorized into `books` and `fishing rods`. Another resource is `cash`. `Cash` is a bulk resource that appears as a dashed drop. Evidently, *Sy's Fish* cannot track each single coin or bill and therefore, only the whole amount of `cash` is of interest. Another bulk resource is `fish`. `Fish` can be categorized into `carp`, `trout`, and `tuna`. It would be quite hard to specify `fish` as a discrete resource, because it would require tagging each individual `fish` with some kind of code. For transporting the products and fish, we define the resource `truck`, which is modeled as a solid

drop. Lastly, there is one specific resource: `labor`. `Labor` defines all the minutes agents can participate (i.e., provide work) in events.

5.7.3 Value Chain View

In the next step a high level overview of *Sy's Fish* business model is provided by depicting it in a value chain view (cf. Figure 5.24). The value chain contains economic activities that create higher value by value transfers with external partners or transformations inside the company. Usually, resources created by one value activity serve as input to another one. Next, we will explain all the value activities which are important for *Sy's Fish* to track.

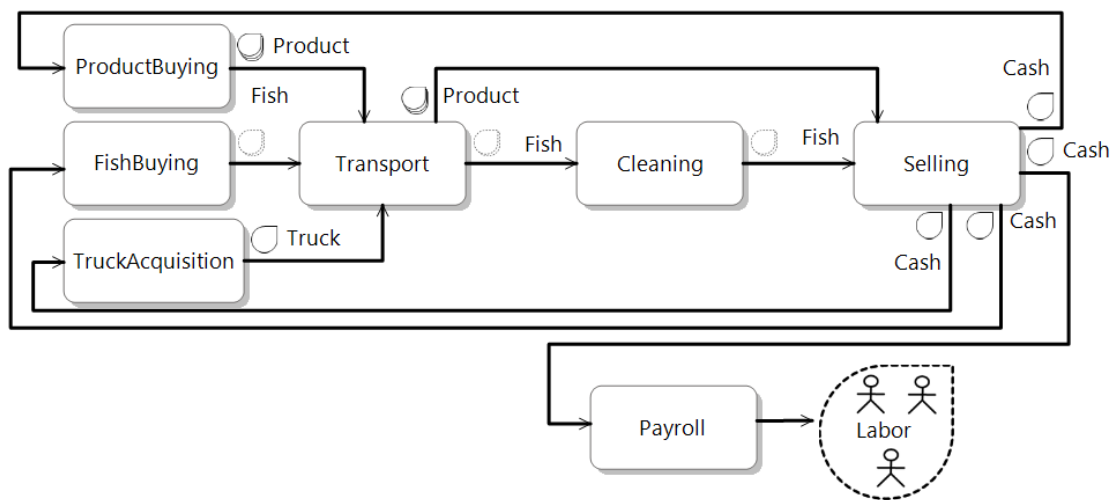


Figure 5.24: Sy's Fish Value Chain View

`Fish` and `products` are purchased with `cash` in the two value activities `fish buying` and `product buying`. The `products` and `fish` are transported to *Sy's Fish* by a truck leased for `cash` in the `truck acquisition` value activity. The `fish` are cleaned in the `cleaning` value activity and then the `products` and `fish` are sold for `cash` in the `selling` value activity. In order to get the resource `labor` from the inside agents, we have to pay them `cash` in the `payroll` value activity. Notice, `labor` is not explicitly depicted as an input to any of the value activities. However, it implicitly serves as an input to all the value activities.

5.7.4 Operational View

Each value activity defined in the value chain view is refined in the operational view. For this example we just elaborate on the `selling` value activity. As we can see in Figure 5.24 for the `selling` value activity, *Sy's Fish* gives up the two resources `product` and `fish` in order to get `cash`. These three resources are also part of the events in the corresponding `selling` duality of the operational view in Figure 5.25. Thus, an automatic generation of a duality stub is provided by the REA-DSL tool.

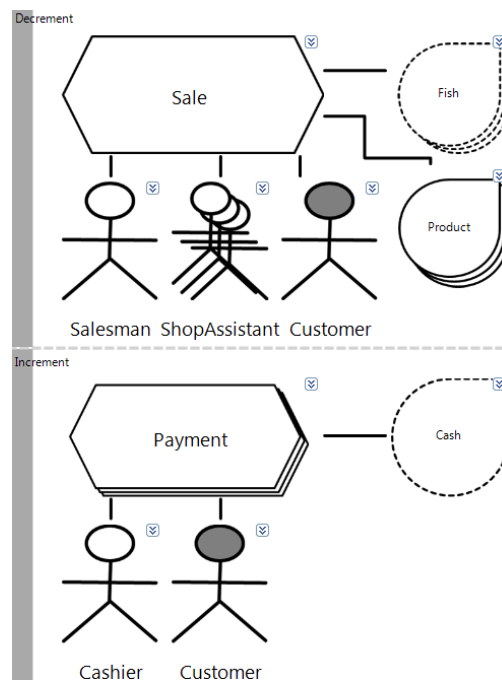


Figure 5.25: Sy's Fish Operational View

The duality shows the give-take principle: *Sy's Fish* sells fish and products in the sale event and therefore gets cash in the payment event. The stack of dashed fish resources means, that multiple fish can be sold at one time (e.g., 10 carp and 5 trout). Again, fish is just a bulk resource, and it therefore cannot be tracked individually. Additionally, many products can be sold in the sale event depicted by the stacked solid product resources. Each of the products has its own unique RFID tag. In a sale event, one salesman is present as well as at least one shop assistant indicated by the agent stack. The outside agent customer receives the fish and products. In order to compensate for the sale event, the customer engages in a payment event where he gives cash to the *Sy's Fish* cashier. Payments are modeled as an event series because payments can be split up into multiple payments over time. Cash must be modeled as a bulk resource, and only the increase of the cash account is tracked.

Notice, there is no labor resource connected through a stock-flow to the sale event. The labor resource is automatically deducted by all the inside participate relationships of the inside agents. This means, whenever inside agents participate in a sale event, the minutes they participate in the event will be subtracted from their labor accounts. In Figure 5.26, all operational views of this example are shown.

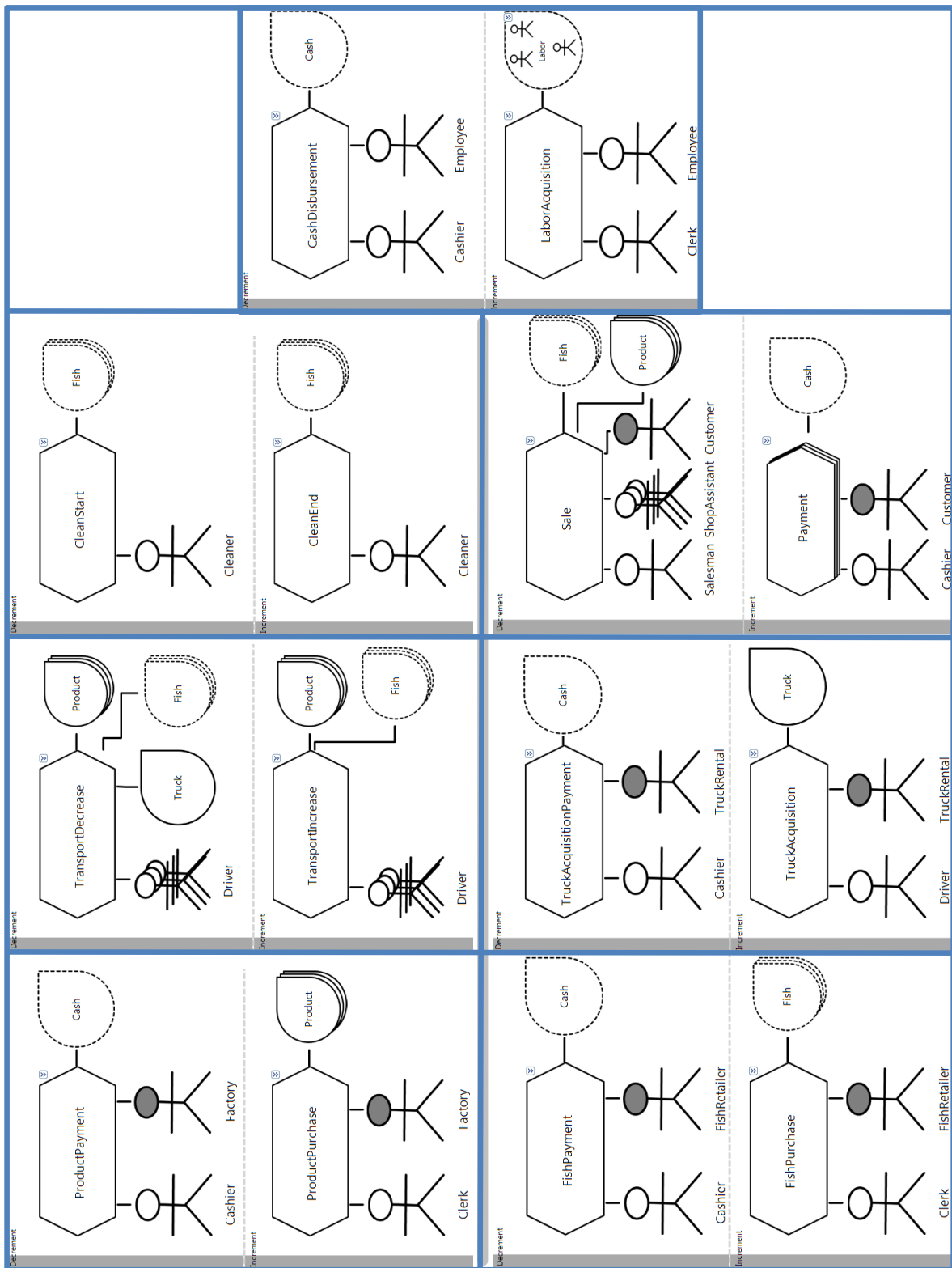


Figure 5.26: REA all Operational Views

5.7.5 Planning View

Each value activity defined in the value chain view can be refined in the planning view. Whether a planning view is modeled or not depends on whether or not *Sy's Fish* wants to record or capture commitments for specific events. For this example, we just elaborate on the *selling* planning view which is depicted in Figure 5.27. The resulting contract is a 2 x 2 matrix. The top covers a commitment leading to a decrease in resources, and the bottom defines the compensating commitment leading to an increase in resources. Orthogonally, the left hand side defines who does the commitment and the right hand side defines what the commitment is about and who is going to fulfill it.

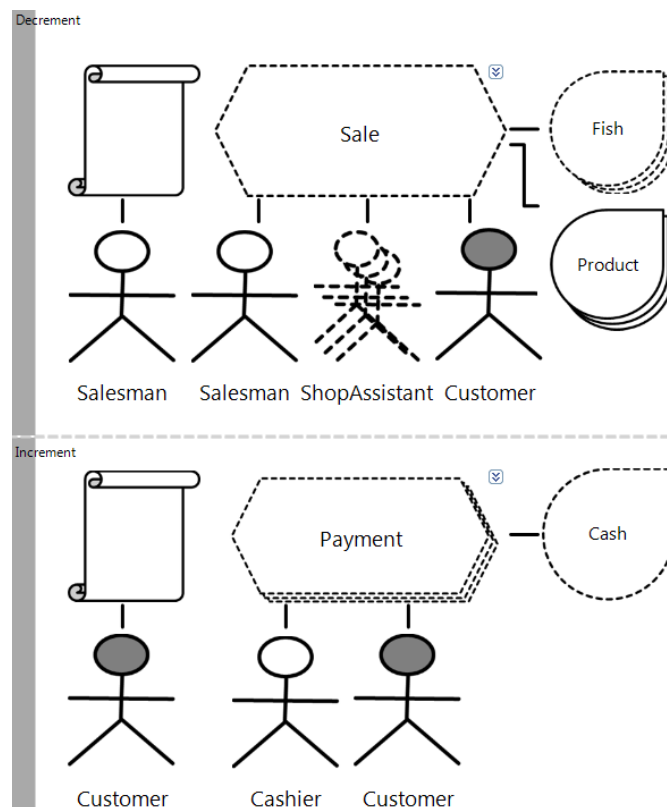


Figure 5.27: Sy's Fish Planning View

The contract is an *order* and contains two commitments: the *salesman* committing in the *sell* commitment to engage in the *sale* event in the future and the *customer* committing in the *pay* commitment to engage in the *payment* events in the future. A concrete *salesman* will participate in the future *sale* of the resources *fish* and *product*. Since more than one *fish* and more than one *product* are potentially sold, their shapes appear as *stacks of drops*. As mentioned before, *fish* are bulk resources with dashed drops and *products* are individually identifiable with *solid drops*. Note, a solid drop in the commitment means also that the individual *product* being part of the future *sale* is exactly defined at the time of the commitment. If only the type of the *product* is specified in the commitment, it has to appear as a *dashed*

drop (or *stack of dashed drops* in case of multiple products). Furthermore, the salesman will be supported by one or more shop assistants (depicted as a stack of dashed stick figures) who are not known at the time of the commitment. These shop assistants will be instantiated when the sale event happens. In the reciprocal commitment, the customer commits to pay in cash to a not-yet-known cashier. The staple of payment events signifies that there is the option of multiple (partial) payments. The contract order we just modeled is executed in the future by the duality selling. One may semi-automatically transfer the right hand side of the contracts in the planning view to the operational view and thus, actually might skip modeling the operational view by hand. This is a big benefit compared to the original REA representation.

Furthermore, policies might be derived from this planning view. These policies would look like this:

- **Policy 1:** a specific sale event type is only allowed to sell specific kind of fish (e.g., Austrian sales are only allowed to sell carp or trout).
- **Policy 2:** a specific sale event is only allowed to sell certain products (e.g., fishing rods and books can be sold in Austrian sales but only books can be sold in US sales).
- **Policy 3:** a certain sale event type requires a certain salesman (e.g., only salesmen speaking English are allowed to participate in an US sale).
- **Policy 4:** only certain shop assistant types might participate in certain sale event types (e.g., only senior shop assistants may assist in an US sale – notice, this requires the definition of additional subtypes of shop assistants).
- **Policy 5:** only certain customers are allowed in certain sale event types (e.g., we only sell fish to big companies in US sales, such as *Bubba Gump* or *Fried Fish Deluxe*).

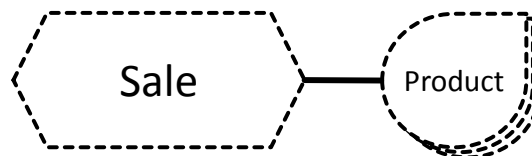


Figure 5.28: Sy’s Fish Planning View Product Alternative

One alternative for modeling the resource product in the sale event type in the contract would be to use stacked dashed product drops as shown in the planning view excerpt in Figure 5.28 instead of the solid product drops as shown in Figure 5.27. The meaning would change from reserving the exact identifiable product at order time to just specifying the amount of certain product types which in the future will be sold and then specified by a concrete identifiable product. An example for this constellation is: When a *fishing rod* of type Big Game Fishing is ordered by the customer, we just commit to sell the specific type of this *fishing rod*. We might have ten fishing rods of type Big Game Fishing on stock. Once we actually sell the *fishing rod*, we take the specific uniquely identifiable *fishing rod* with the RFID XYZ123 and give it to the customer. In Figure 5.29 all planning views of this example are shown.

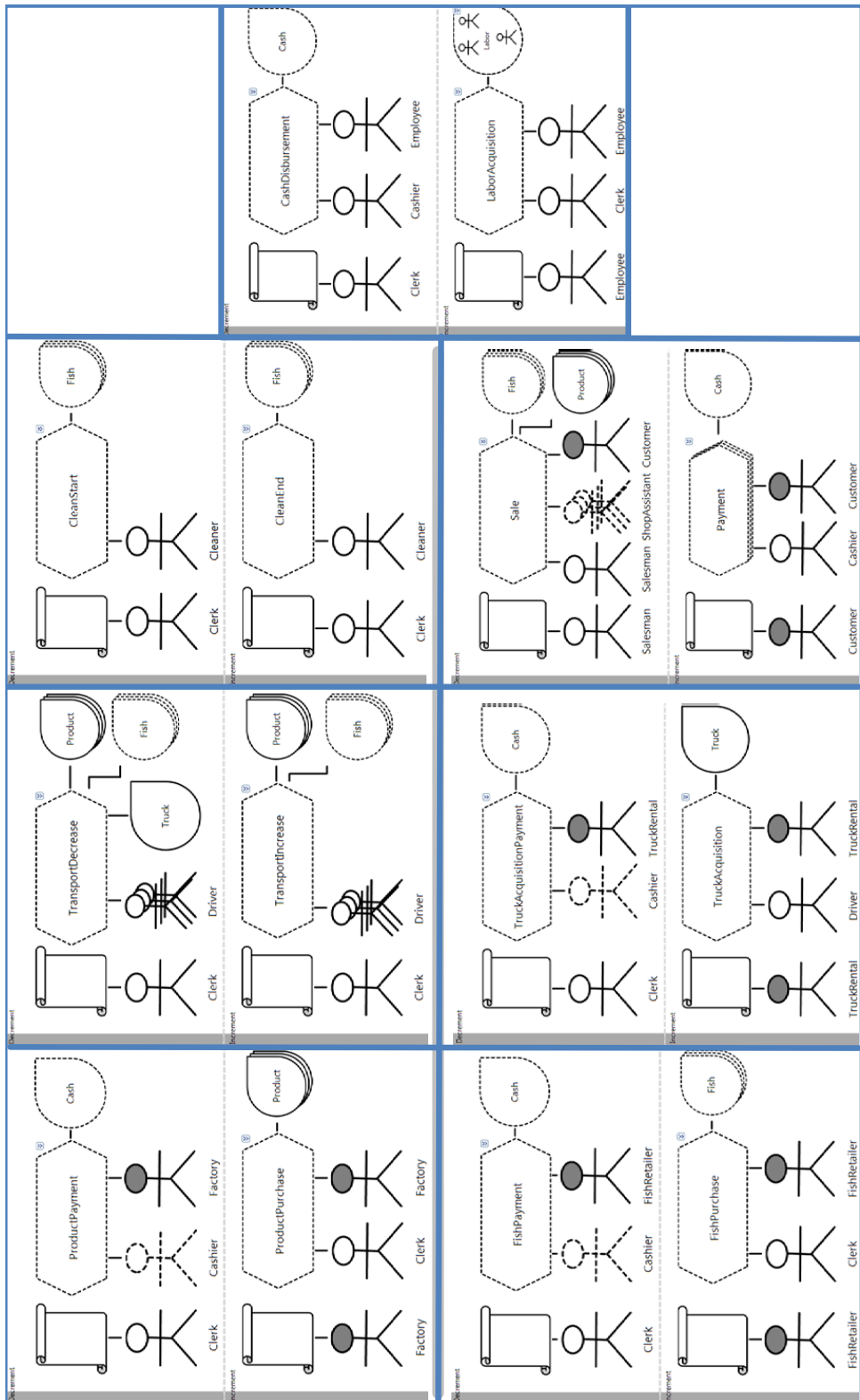


Figure 5.29: REA Example all Planning Views

REA-DB

In the previous chapters we took the challenge to create an unambiguous and easy to understand domain-specific language for REA called the REA-DSL. It is formalized by an underlying meta-model and provides an intuitive graphical representation with different shapes for different REA concepts. However, it still misses some concepts indispensable for an AIS database scheme – i.e., properties and primary keys. Properties can be defined on resources, agents, events, commitments, stockflows, and participate relationships on the different REA layers. Policy properties on stockflows and participate relationships allow us to define policy relationships between events, agents, and resources. In the first part of this chapter we extend the REA-DSL by properties and primary keys followed by the mapping to the relational model.

After adding primary keys and properties, the REA-DSL contains all information needed for describing a relational database schema for an AIS. Business professionals and IT professionals can together use the REA-DSL tool to model the requirements for the conceptual model of an AIS. Nevertheless, these models cannot be read and processed by a database system the AIS is based on. The IT professional would still be required to remodel the REA-DSL to a relational model adhering to the REA ontology rules. This task is error prone and time intensive – resulting in additional costs for designing the AIS. Consequently, in the second part of this chapter we present a mapping between the REA-DSL and a relation model. Additionally, our REA-DSL tool provides an automatic generation of a relational model from the REA-DSL based on the proposed mapping. We argue, that our REA-DSL approach to model an AIS better reflects the real world economic phenomena of a company and creates more accurate data structures according to the REA ontology than the class-like representation does. Furthermore, we believe that it saves a lot of time during the design process of an AIS by providing proper tool support.

To extend the REA-DSL with attributes and identifying keys we make use of common modeling practice in Entity-Relationship (ER) modeling and refer to basic database literature [Dat03, EN10, PU03, Vos08]. As for the mapping from the REA-DSL to the relational model we also consulted the aforementioned books as well as the hierarchy mapping strategies in [CC05].

In the following we introduce the extension of the REA-DSL by properties and primary keys. Afterwards, the main part of the chapter elaborates on the mapping rules between the REA-DSL and the relational model.

6.1 REA-DSL Extended: Adding Properties and Keys

In order to enable the REA-DSL to derive relational models we extend the main REA-DSL concepts agents, resources, events, stockflows, participate relationships, and commitments by properties and primary keys. We explain the properties and primary keys on agents in detail and follow with resources, events, stockflows, and participate relationships. To visually assist the modeler, properties affecting the operational layer are colored in green, properties affecting the planning layer are colored in purple, and properties affecting policy layer are colored in yellow. This color code is also used by Bill McCarthy in class-like REA examples.

Agent Properties. In Figure 6.1 on the left side, the properties are depicted in the concrete syntax of the agent REA-DSL and on the right side the extended parts of the agents meta-model is shown. The arrows annotated by numbers in circles relate the concrete syntax of the REA-DSL to the corresponding meta-model element. An agent (e.g., `employee`) is depicted by a stick figure (1) and can contain multiple *properties* in a green compartment called *object properties* (3). Properties consist of a *name* and a *type*. In the concrete syntax on the left side the name of the property is separated from the type of the property by a colon (e.g., `Age : INT`). The type can usually be an SQL type. In our tool, we provide the following subset of SQL types: BOOLEAN (true or false), DATE, DATETIME, DOUBLE, INT, TIME, TIMESTAMP, VARCHAR (text). The flag *isPrimaryKey* specifies, whether or not this property is used as (part of) the unique identifier of the agent. Primary key properties are tagged by a key symbol (e.g., `EmployeeId : INT`).

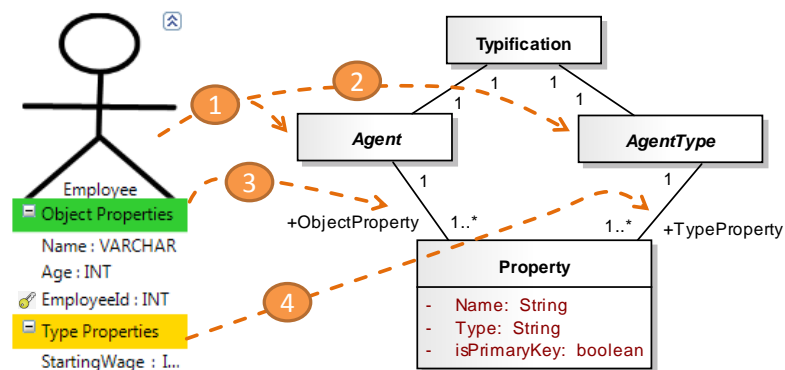


Figure 6.1: Agent Properties Meta-Model

The instances of the object properties vary between each individual employee. However, there are also properties which stay the same for certain types of agents called *type properties* contained in a yellow compartment (4). Thus, REA distinguishes between *agents* with object properties and *agent types* with type properties. For example, all employee types *driver* (or all employee types *cashier*) have the same common *starting wage*. Therefore, the meta-model specifies, that one agent has exactly one *typification* relationship to exactly one *agent type* and vice versa. Consequently, there is a one-to-one relationship between the agent and the agent

type which are depicted by one common stick figure in the concrete syntax (1,2). In the object-oriented paradigm classes may comprise of static attributes (class attributes) and non-static (regular) attributes. In the REA context it is required to have one concept covering the regular attributes (*object properties*) and another one covering the static attributes (*type properties*). For example, a commitment might just reference the agent type (e.g., employee type driver) containing the type properties or the exact individual agent (e.g., driver John) containing the object properties.

Resource Properties. Figure 6.2 shows the resource view concrete syntax and the extended parts of the resource view meta-model. Resource properties depicted on the left side of Figure 6.2 are quite similar to the agent properties explained above. Consequently, a resource (1) also contains two compartments: an object properties compartment colored green (2) and a type properties compartment colored yellow (3). In our example, the resource `product` contains two object properties: `RFID:INT` and `Name:VARCHAR`. Since the `RFID:INT` property is marked by a key symbol, it will be used as the unique identifier for the `product` table. The type property `Tax:INT` applies to a product type. Consequently, a certain type of product (e.g., books) will always have the same tax applied on it (e.g., 7%).

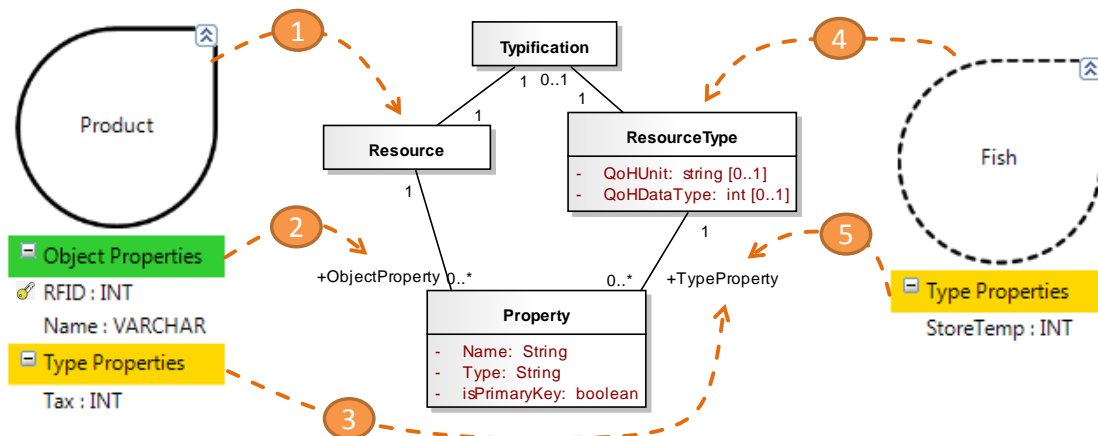


Figure 6.2: Resource/Bulk Resource Properties Meta-Model

On the right side of Figure 6.2 a bulk resource (resource type) `fish` (4) is depicted. As previously described, a bulk resource cannot be individually identified. In our `fish` case, we can only record the amount of a type of `fish` but cannot track each individual `fish`. Thus, it is not possible to apply object properties for an individual `fish`. Consequently, bulk resources may only contain type properties (5). The `fish` bulk resource contains the type property `StoreTemp:INT`. Thus, we can record for each type of `fish` the storage temperature (e.g., tuna 35 degrees Fahrenheit, carp 40 degrees Fahrenheit). Additionally, a bulk resource (resource type) contains two additional fixed properties: `QoHUnit` (Quantity on Hand Unit) as a string and `QoHDataType` (Quantity on Hand Data Type) as an integer. `QoHUnit` defines the

unit, the sum of all the resources is measured in (e.g., **pounds** as in 300 pounds of tuna). `QoHDataType` defines the data type of the unit (e.g., data type **double** for **pounds**).

Event and Commitment Properties. The left side of Figure 6.3 depicts a `sale` event (1) on the operational layer. Events can have multiple event properties (2) colored green. In the `sale` event example we have one object property `SaleNr:INT` and one object property `SaleDate:Date`. These properties will be set for each individual sale once it occurs. Each sale can be uniquely identified by the primary key `SaleNr` specified by the primary key symbol. On the right side you can see a `sale` event type (3) on the planning layer. The event type contains an event properties compartment (4) colored green, an event type properties compartment (5) colored yellow, and a commitment properties compartment (6) colored purple.

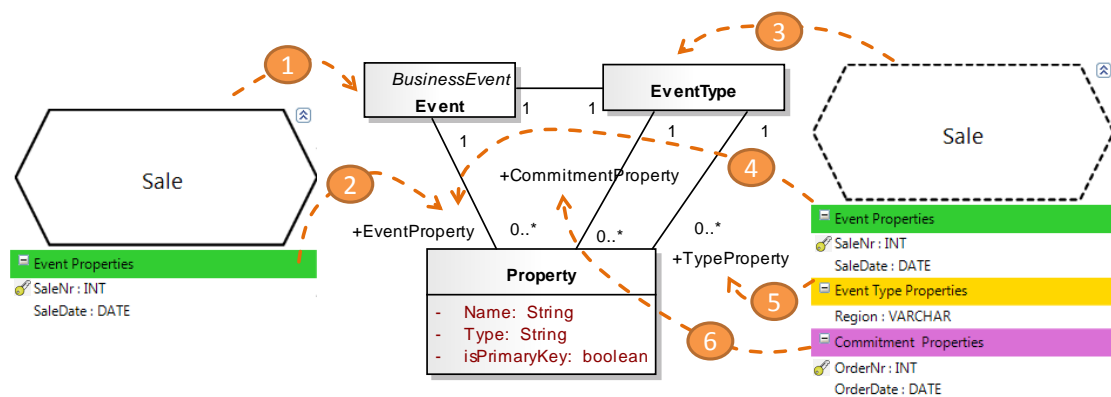


Figure 6.3: Event/Event Type Properties Meta-Model

You may notice, that we already defined the event properties compartment for the event on the operational layer. As mentioned earlier, if the planning layer is conceptually congruent to the operational layer, the operational layer is implicitly covered by the planning layer. Thus, we also provide for the planning view the same event properties compartment (4) as for the operational view (2). However, if the operational view is not conceptually congruent to the planning view, the object properties have to be defined in a separate operational view. Event type properties (5) define properties for a special kind of event. In our `sale` event type example the event type property is `Region:VARCHAR` (e.g., Austrian sale, US sale). Thus, for example an actual `sale` with the `SaleNr` SA341 can be of the `sale` type Austrian sale. Furthermore, we can define properties for commitments made in advanced for this sale event. These commitment properties are contained in the commitment properties compartment (6). According to our example, we define the two commitment properties `OrderNr:INT` and `OrderDate:DATE`; `OrderNr` is defined as the primary key by the key symbol.

Stockflow and Participate Properties. Properties might as well be specified for stockflows between events and resources, or on participate relationships between events and agents (cf. Figure 6.4). However, properties for stockflows and participate relationships do not require a

primary key flag. We start by explaining properties on stockflows and participate relationships on the operational layer depicted on the bottom of Figure 6.4.

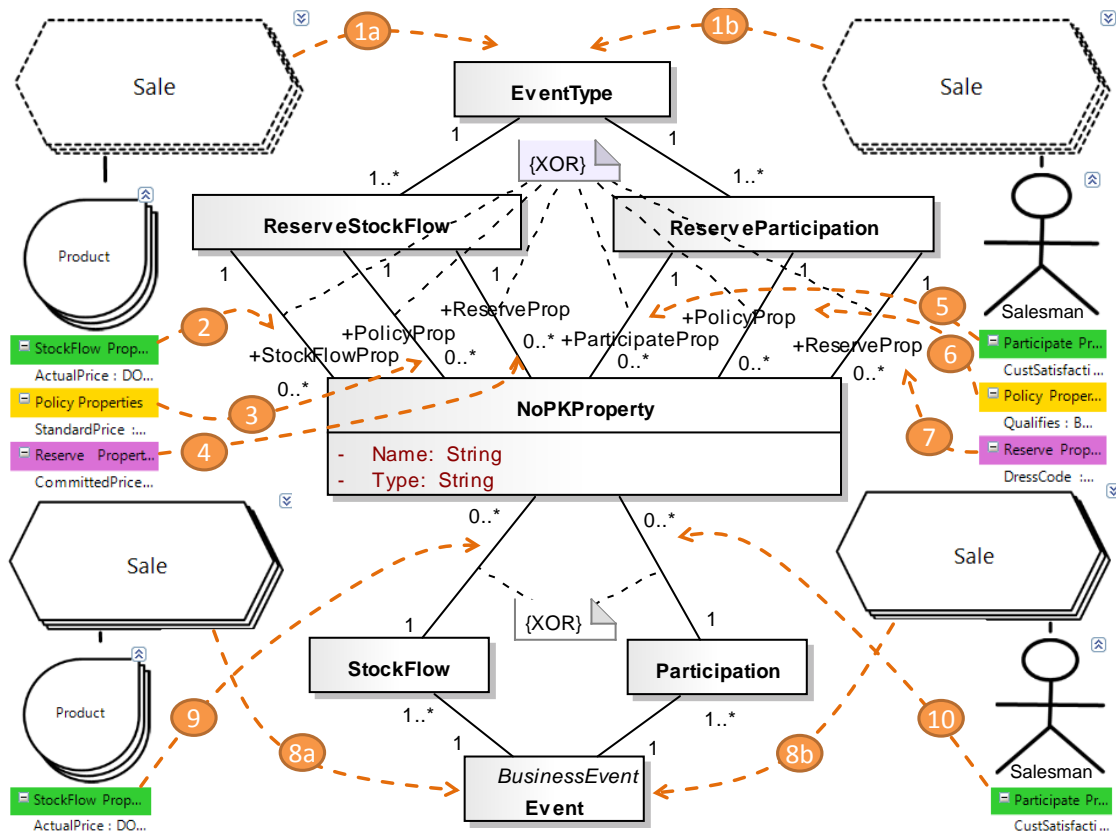


Figure 6.4: Stockflow and Participate Properties Meta-Model

On the bottom left you can see a stockflow between the event series `sale` (8a) and the resource series `product` (notice, it does not matter whether we talk about resource series and event series or resources and events; there is no difference in the meaning of the properties of the stockflow and the participate relationship). A stockflow contains a stockflow properties compartment (9) colored in green. These are properties important for the actual stockflow happening on the operational layer. In our example we record the actual price paid for a product in a sale event as indicated by the `ActualPrice:DOUBLE` property. Since these properties are added to an existing table with existing primary keys, we do not need to define primary keys for stockflows and participate relationships. Consequently, the properties changed from the meta-model class `Property` to the meta-model class `NoPKProperty` which does not contain an `isPrimaryKey` attribute.

On the bottom right of Figure 6.4 you can see a participate relationship between the sale event series (8b) and the salesman agent. The participate relationship contains a participate properties compartment (10) colored in green. It stores properties for the actual participation in an event on the operational layer. In our `sale` example we store the customer

satisfaction for each agent in each individual sale by the `CustSatisfaction:VARCHAR` property.

The stockflows and participate relationships of the planning layer are depicted on the upper half of Figure 6.4. Similar to the properties of events explained earlier, stockflow properties and participate properties (green compartments) from the operational layer can also be defined on the planning layer in case the events are conceptually congruent (2,5).

On the upper left a reserve stockflow between a `sale` event type series (1a) and a resource series `product` is depicted. Additionally to the stockflow properties (2,9) on the operational layer, it now contains a policy properties compartment (3) colored in yellow and a reserve properties compartment (4) colored in purple. The policy properties (3) allow us to define properties for policy relationships between resource types and event types. In our case we define the policy property `StandardPrice:DOUBLE`. This for example allows us to define an instance on the policy layer that specifies, that usually for an `Austrian Sale` the standard price for a specific type of book (`cooking fish right`) is 9.90 dollars, whereas, for a `US Sale`, the price of the same book type is 7.90 dollars.

You might want to define special properties for a stockflow of an order made in advance for the actual sale of a product. Therefore, you may define properties in the reserve properties compartment (4). The `CommittedPrice:DOUBLE` specifies the committed price for a product in a certain future sale. For example, in an order for the book `cooking fish right` we give a special discount and commit to sell the book for 6.90 dollars instead of the standard 7.90 dollars defined in the policy before. Two days later the customer comes to acquire the ordered book. He notices a stain on the cover and we therefore provide an additional discount of one dollar. Thus, the actual price (2) paid in the sale was 5.90 dollars. This example shows nicely, how the prices on the different layers (policy, planning, and operational) might vary.

The upper right of Figure 6.4 shows a reserve participate relationship between the `sale` event type series (1b) and an agent `salesman` on the planning layer. It specifies, that a certain salesman is reserved for a special sale in the future by an `order`. Additionally to the participate properties (5,10) on the operational layer, the participate relationship also contains a policy properties compartment (6) colored yellow and a reserve properties compartment (7) colored purple. Thus, we are able to define properties on policies between event types and agents. In our example the `Qualifies:Boolean` property specifies, which salesman agents are qualified for which `sale` event types. For example, because `Harry` knows German, he qualifies to participate in `Austrian sale` event types. Furthermore, you might want to apply special properties on a participate relationship for a future `sale` event in an `order` commitment. Thus, for an event type you might define reserve properties (7) on the participate relationship. In our example, when an order comes in, you want to define special `clothes` a salesman has to wear on the day of the actual sale (e.g., if we have an order from our royal customers, the salesman has to wear a business suite, otherwise he might just wear a t-shirt). Therefore, we define the reserve property `DressCode:VARCHAR` to let the salesman know in advance what he has to wear.

6.2 REA-DSL to Relational Model Mapping

In this section we create a model mapping between the REA-DSL and a relational model to automatically generate a relational model from the REA-DSL models in terms of SQL statements. We decided to map to the relational model directly instead of an Entity-Relationship diagram or a class diagram. First of all, this gives us full generation control of the database data structure. The idea is, that in a latter stage, the mapping and appearance of the relational model could be fine tuned by flags and preferences set by the IT professional. Second, the generated SQL statements are widely accepted and can be imported by many database systems and database modeling tools. In our case we used the free MySQL Workbench [Ora11] to import and show the relational model diagrams.

For educational purposes, we believe that the mapping is best explained by using the concrete syntax of the REA-DSL and the relational model. The exact mapping is described by a pseudo-code. The mapping of the REA views agent view, resource view, operational view, and planning view is explained in the following. As for the fifth view – the value chain view – there is no mapping provided since it is implicitly covered by the transformations of the operational views.

General Mapping Rules. Agents and resources can have generalization relationships in REA. In this example these generalization hierarchies have a maximum depth of two and are considered to be complete and disjoint which is the most common case in REA. For mapping the *generalization hierarchies* of the REA-DSL there are several common approaches to a relational model in literature [PU03, Vos08, CC05]. We decided to create a separate table for each super and sub class where the sub class table references the super class table. This allows us to either directly reference the super or sub class which is needed by some REA concepts such as *events* or *commitments*.

For all REA concepts with properties (i.e., agent, resource, event, and commitments) the object properties (green compartments), commitment properties (purple compartments, and type properties (yellow compartments) of the REA concepts always become columns of the corresponding tables. Additionally, properties marked as *isPrimaryKey* and depicted by a key symbol become the primary key column of the table. If there is no primary key defined for type properties (yellow compartment), a *typename* column which serves as the primary key will be created.

6.2.1 Agent View Mapping

On the left side of Figure 6.5 an example of the *Agent View* is depicted. It shows the super agent Employee, which can either be a Cashier or a Driver. On the right side, the corresponding relational model is shown. The mapping between the *Agent View* and the relational model is indicated by the dashed arrows annotated by numbers between the models. The exact agent mapping rules are described by the pseudo code in Algorithm 6.1. A primary key column is indicated by a "P" in a circle and a foreign key column by an "F" in a circle. The mapping explained in the following using inside agents can be applied in the same way on outside agents. In a first step, we need to create a table for the possible agent types. Thus, each *super agent* (Employee) is mapped (2) to an agent type table (AT_EmployeeType) with type property columns (StartingWage). Additionally, an agent type table gets a *TypeName* column which also becomes the primary key if no specific primary key was specified in the type properties. The

sub agents of the generalization hierarchy on the left actually specify, which types of employees are allowed. Thus, the names of the sub agents get inserted into the agent type table, in our case the sub agents Cashier and Driver.

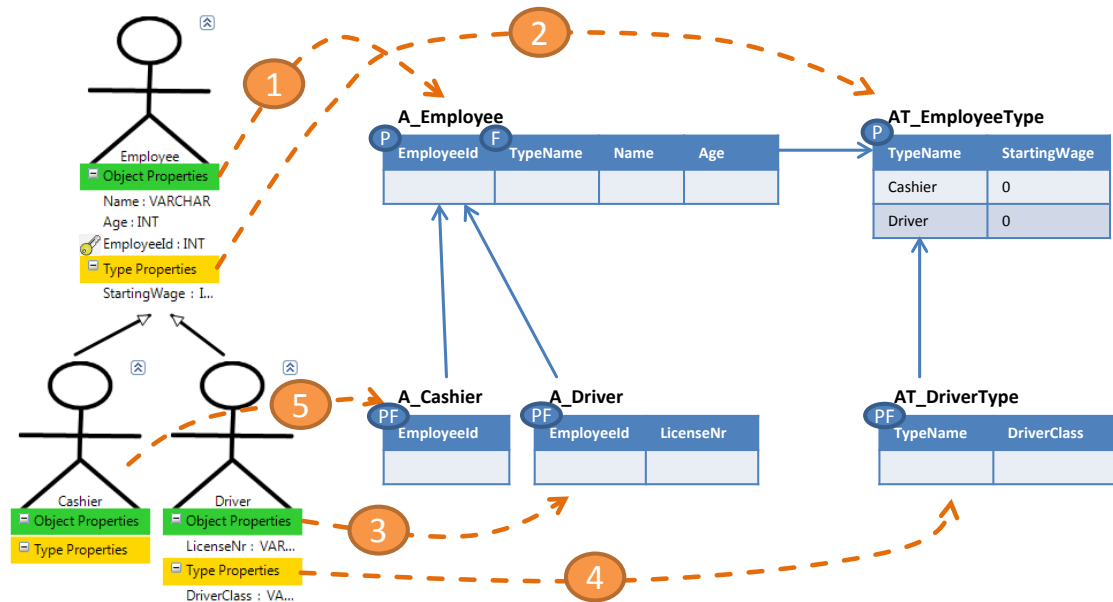


Figure 6.5: Agent Mapping

```

1: for all Super Agents do
2:   create Agent Type Table with the name <Agent.Name>Type;
3:   - add column TypeName of type VARCHAR and make it the primary key;
4:   - add all Type Properties as columns with the specified type;
5:   - insert the name of each related Sub Agent into the table;
6:   create Agent Table with the name <Agent.Name>;
7:   - add all Object Properties as columns with the specified type;
8:   - make the primary key object property the primary key column;
9:   - create foreign key to the Agent Type Table;
10: end for
11: for all Sub Agents do
12:   if Sub Agent has Type Properties then
13:     create Agent Type Table with the name <Agent.Name>Type;
14:     - add column TypeName of type VARCHAR and make it the primary key;
15:     - make column TypeName a foreign key to its Super Agent Type Table;
16:     - add all Type Properties as columns with the specified type;
17:   end if
18:   if Sub Agent has Object Properties or generate all sub agent tables is true then
19:     create Agent Table with the name <Agent.Name>;
20:     - add all Object Properties as columns with the specified type;
21:     - create same primary key as in Super Agent and make it a foreign key;
22:   end if
23: end for

```

Algorithm 6.1: Agent Mapping Rules

In a second step, we create for each *super agent* an agent table (1) which holds all individual agents. In our case we create an `A_Employee` table with all the object properties `Name`, `Age`, and `EmployeeId` (primary key) as columns. An additional column `TypeName` references the agent type table `AT_EmployeeType`. Thus, an `A_Employee` row entry can either be an `AT_EmployeeType` `Cashier` or `Driver`.

Each of the sub agents (`Cashier` and `Driver`) become their own agent table (`A_Cashier` and `A_Driver`) with the object properties columns (`LicenseNr` for `A_Driver`) (3,5) regardless of having specific properties or not. Consequently, each row entry for `A_Cashier` or `A_Driver` also has to have a row entry in `A_Employee`. They reference their super agent (`Employee`) with the primary key `EmployeeId`. If a sub agent (`Driver`) also has type properties, a specific agent type table (`AT_DriverType`) referencing the super agent table (`AT_EmployeeType`) is created (4). Note, there is no reference between `A_Driver` and `AT_DriverType` since it is implicitly referenced through `A_Employee` and `AT_EmployeeType`.

6.2.2 Resource View Mapping

The resource mapping (cf. Figure 6.6 and the pseudo code in Algorithm 6.2) is similar to the agent mapping. In the following, we explain the mapping of regular resources, bulk resources, and labor resources.

Regular Resources. The super resource (`Product`) becomes (2) a resource type table (`RT_ProductType`) with the type property column `Tax` and the primary key column `TypeName`. The sub resources `Book` and `FishingRod` become (3,4) row entries. Furthermore `Product` also becomes (1) a resource table `R_Product` with the object property columns `RFID` and `Name` as well as a `TypeName` column referencing the resource type table `RT_ProductType`. Each sub resource (`Book` and `FishingRod`) also become a separate agent table (`R_Book` with columns `Used` and `ISBN`, and `R_FishingRod` with column `Length`). If a sub resource has type properties (`Book`) (5), a separate resource type table (`RT_BookType` with column `Genre`) referencing the super resource type table (`RT_ProductType`) is generated.

Bulk Resources. In the REA-DSL there are special resources called bulk resources depicted by a dashed drop. These bulk resources are not individually identifiable. Thus, only the resource type and its quantity on hand (QoH) is recorded. `Fish` is such a bulk resource, where we just record the amount of fish we have. According to the mapping in Figure 6.7 a bulk resource can just have type properties and no object properties. The super bulk resource `Fish` is mapped to the bulk resource type table `RB_FishType` including the type properties as columns (`StoreTemp`) and the primary key column `TypeName`. Additionally, a quantity on hand (QoH) column for storing the amount of the bulk resource is created. All the sub bulk resources get inserted as row entries into the super bulk resource table (`RB_FishType`). If a sub bulk resource has specific type properties (e.g., `CarpFamily` in `Carp`), a separate bulk resource table (`RB_CarpType` with `CarpFamily` column) is created referencing the super bulk resource table (`RB_FishType`).

Labor Resource. A special kind of resource in REA is the Labor Resource which is mapped individually (cf. Algorithm 6.3). Labor records the available minutes of labor for using a specific employee. Labor is increased due to the payment event and decreased

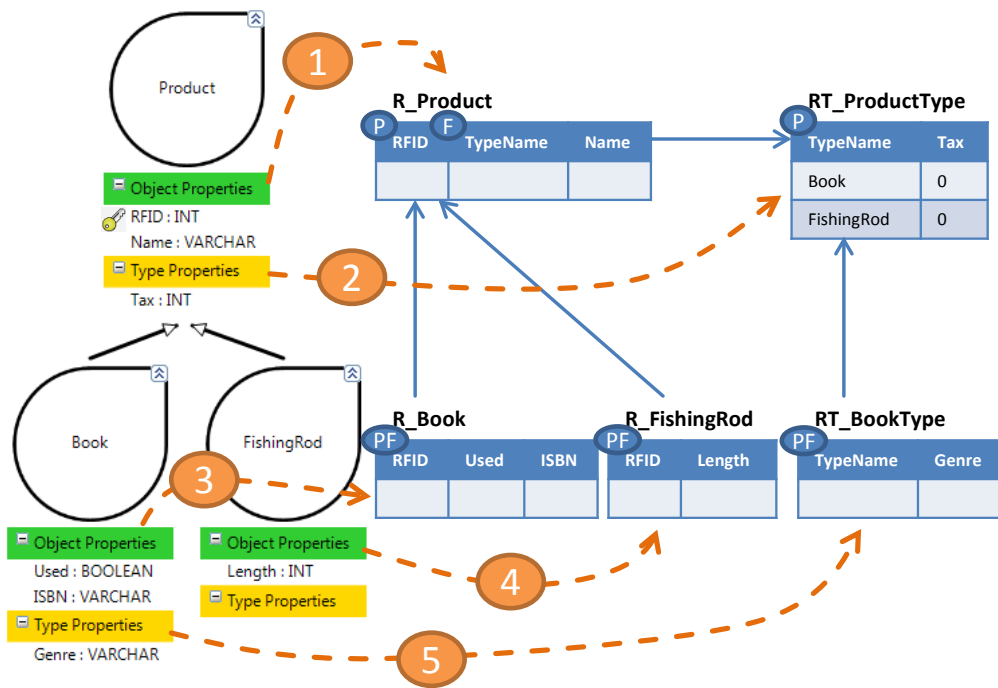


Figure 6.6: Resource Mapping

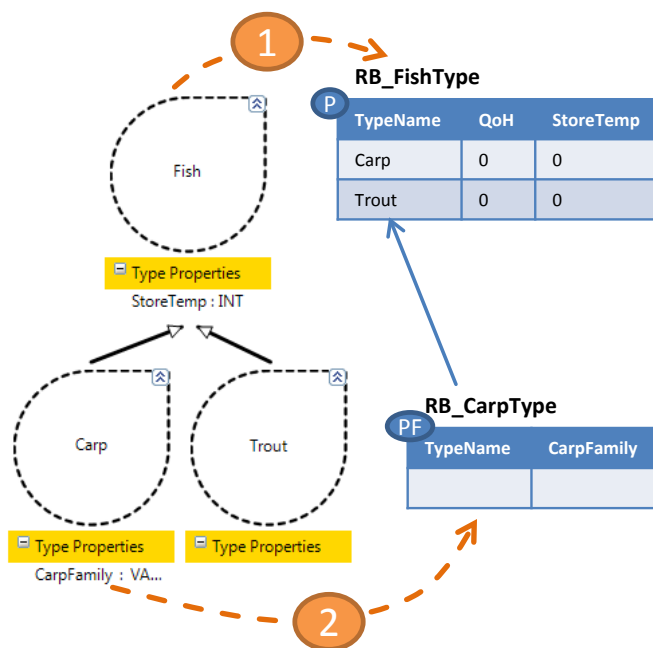


Figure 6.7: Bulk Resource Mapping

```

1: for all Super Resources do
2:   create Resource Type Table with the name <Resource.Name>Type;
3:   - add column TypeName of type VARCHAR and make it the primary key;
4:   - add all Type Properties as columns with the specified type;
5:   - insert the name of each related Sub Resource into the table;
6:   if Super Resource is a Bulk Resource then
7:     - add column QoH with data type QoHDataType;
8:   end if
9:   if Super Resource has Object Properties then
10:    create Resource Table with the name <Resource.Name>;
11:    - add all Object Properties as columns with the specified type;
12:    - make the primary key object property the primary key column;
13:    - create foreign key to the Resource Type Table;
14:   end if
15: end for
16: for all Sub Resources do
17:   if Sub Resource has Type Properties then
18:     create Resource Type Table with the name <Resource.Name>Type;
19:     - add column TypeName of type VARCHAR and make it the primary key;
20:     - make column TypeName a foreign key to its Super Resource Type Table;
21:     - add all Type Properties as columns with the specified type;
22:   end if
23:   if Sub Resource has Object Properties or generate all sub resource tables is true then
24:     create Resource Table with the name <Resource.Name>;
25:     - add all Object Properties as columns with the specified type;
26:     - create same primary key as in Super Resource and make it a foreign key to it;
27:   end if
28: end for

```

Algorithm 6.2: Resource Mapping Rules

by agents participating in events (e.g., John drives the truck for 120 minutes). Consequently (cf. Figure 6.8), Labor is mapped to a bulk resource table (RB_Labor) with a primary key column TypeName and an EmployeeId column referencing the corresponding employee in the A_Employee table. The column QoHInMinutes stores the minutes available for an employee.

Consequently, in the Labor Acquisition event the Labor resource of the specific participating agent will gain minutes. Thus, the Labor Acquisition event table has to reference the Labor resource table. This works similar to all other events and resources. However, consuming the resource Labor works differently. Labor is consumed when inside agents participate in events. Through the foreign key EmployeeId of the Labor resource to the inside agent, the minutes of the Labor resource of the participating inside agent can be referenced and hence, updated. For example, if a specific inside agent was involved in a certain event for one hour, 60 minutes will be subtracted from his labor record in the labor table.

```

1: for Labor Resource do
2:   create Labor Resource Type Table with the name RB_Labor;
3:   - add column TypeName of type VARCHAR and make it the primary key;
4:   - add all Type Properties as columns with the specified type;
5:   - add column QoHInMinutes with data type INTEGER;
6:   - add column Super Inside Agent Primary Key with the corresponding data type to reference the
   inside agent, who the minutes of labor belong to;
7: end for

```

Algorithm 6.3: Labor Resource Mapping Rules

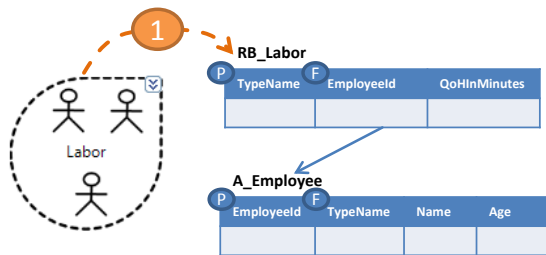


Figure 6.8: Labor Resource Mapping

6.2.3 Operational View Mapping

The REA building blocks agent and resource are used in the operational view (duality) which defines what resources are exchanged by which agents in which economic events. An operational view exists for each value activity in the value chain view. As an example for the mapping we take the operational view *Selling* duality used for the example in the REA-DSL chapter in Figure 5.25. The exact operational mapping rules are described by the pseudo code in Algorithm 6.4.

Duality and Event. As shown in Figure 6.9 the *Selling* duality (1) is mapped to a duality table `D_SellingDualityTransfer` with the primary key `DualityId`. If a contract is specified for this duality (in the planning view), it also references this contract by the `ContractId` column. Each event of the duality is mapped to an event table. For the *Sale* event (2) an event table `E_SaleEvent` is created with all the event properties (`SaleNr` and `SaleDate`) as columns and the `SaleNr` as primary key. Furthermore the `SellingDualityTransferId` column references the duality it belongs to and `SaleEventType` references the event type of the *Sale* event (generated in the planning mapping). If a commitment is defined for this event in the planning view a `SaleCommitmentOrderNr` is created referencing the corresponding commitment this event fulfills.

Agents. Single participating agents (*Salesman* and *Customer*) are referenced directly from the event table (`SalesmanEmployeeId` and `CustomerCustomerId`). Additionally, for inside agents (*Salesman*) we also record the minutes used in a separate column (`SalesmanMinutesUsed`). These minutes used will be subtracted from the bulk resource table *Labor* referencing the corresponding employee *Salesman*. Multiple participating agents (*ShopAssistant*) become (5) their own participation table `E-A_SaleEventParticipateShopAssistant`. The table has columns for referencing the *SaleEvent* and the *ShopAssistant* as well as the `MinutesUsed`.

Resources. Multiple resources (*Product*) become (4) their own stock-flow table (`E-R_SaleEventStockFlowProduct`) referencing the corresponding *SaleEvent* and the individual *Product*. Multiple bulk resources *Fish* also become (3) their own stock-flow table (`E-RB_SaleEventStockFlowFishType`) referencing the corresponding *SaleEvent* and the *FishType*. Additionally, for bulk resources a quantity column is created (`FishTypeQuantity`). Notice, resources can be marked as being used instead of being exchanged. In such a case, also an additional column *minutes used* is generated.

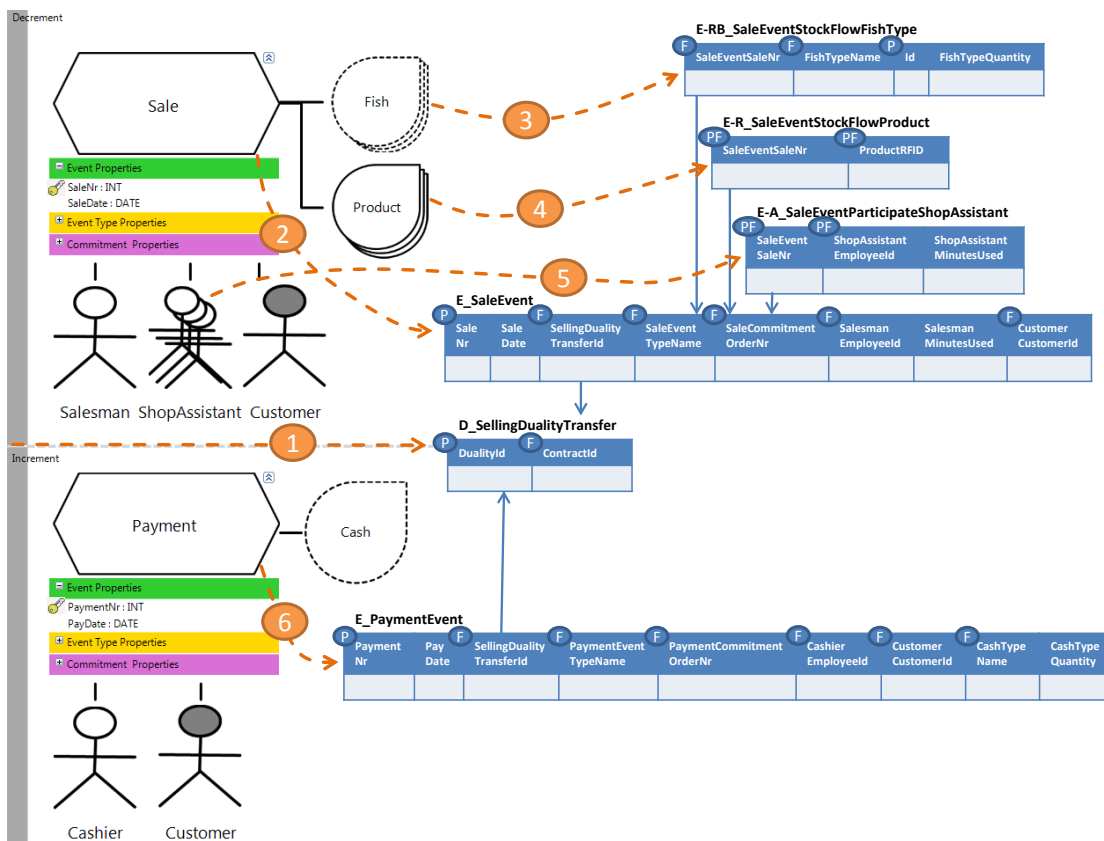


Figure 6.9: Operational Mapping

The Payment event is mapped (6) to an event table (`E_PaymentEvent`) in a similar way as the Sale event. As an additional concept, the single bulk resource `Cash` is directly added as column `CashTypeName` in the payment event table referencing the `CashType` and a `CashTypeQuantity` column storing the amount of `Cash` being exchanged.

Operational Stockflow and Participation Properties Mapping. As depicted in Figure 6.10 stockflows with resources and participation relationships with agents might also have stockflow properties and participation properties, respectively. The rules applied for the transformation to the relational model are shown in Algorithm 6.5.

Stockflow Properties. According to our example, the `Product` resource series has a stockflow property `ActualPrice:DOUBLE`. Since it is a resource series, a separate `E-R_SaleEventStockFlowProduct` table is created by the operational mapping rules mentioned previously. An additional column called `ActualPrice` of the data type `DOUBLE` (2) is now added to this table. If the `Product` resource series is modeled as a single `Product` resource instead, the location of the created column changes. In that case, the stockflow properties become columns in the `E_SaleEvent` table.

```

1: create Duality Table with the name <PlanningModel.Name>DualityTransfer/Transformation;
2: - table name either Transfer or Transformation according to model type;
3: - add DualityId as primary key column;
4: - add foreign key ContractId to the ContractTable;
5: for all Events do
6:   create an Event Table;
7:   - add all Object Properties as columns with the specified type;
8:   - make the primary key object property the primary key column;
9:   - add foreign key to the Duality Table and the Event Type Table;
10:  - add foreign key to the Commitment Table which this event fulfills;
11:  for all Connected Agents (participations) do
12:    if Single Agent then
13:      foreign key to Agent Table and add minutes used if is inside agent;
14:    else if Multiple Agent then
15:      create a EventParticipateAgents Table;
16:      - foreign key to Event Table and Agent Table;
17:      - add minutes used column if is inside agent;
18:    end if
19:  end for
20:  for all Connected Resources (stock-flows) do
21:    if Single Resource then
22:      foreign key to Resource Table and add minutes used if isUsed;
23:    else if Multiple Resource then
24:      create a EventStockFlowResources Table;
25:      - foreign key to Event Table and Resource Table;
26:      - add minutes used column if isUsed;
27:    else if Single Bulk Resource then
28:      foreign key to Resource Type Table and add minutes used if isUsed;
29:      add quantity (of quantity type) column;
30:    else if Multiple Bulk Resources then
31:      create a EventStockFlowResourceTypes Table;
32:      - add Id as primary key column;
33:      - foreign key to Event Table and Resource Type Table;
34:      - add quantity (of quantity type) column;
35:      - add minutes used column if isUsed;
36:    end if
37:  end for
38: end for

```

Algorithm 6.4: Operational Mapping Rules

Participate Properties. Next, we have a participate relationship of the Salesman inside agent to the Sale Event. This participation relationship has an additional property `CustSatisfaction:VARCHAR`. Since the agent is a single agent, the agent reference is part of the event table `E_SaleEvent` (1). Consequently, we also add an additional column `CustSatisfaction` of the data type `VARCHAR` (3) to the `E_SaleEvent` table. Notice, we do not show all the columns of the `E_SaleEvent` table here to keep the figure smaller for the example. If the Salesman agent is modeled as an agent series instead, the location of the created column changes. In that case, the participate properties also become columns in a separate participate table.

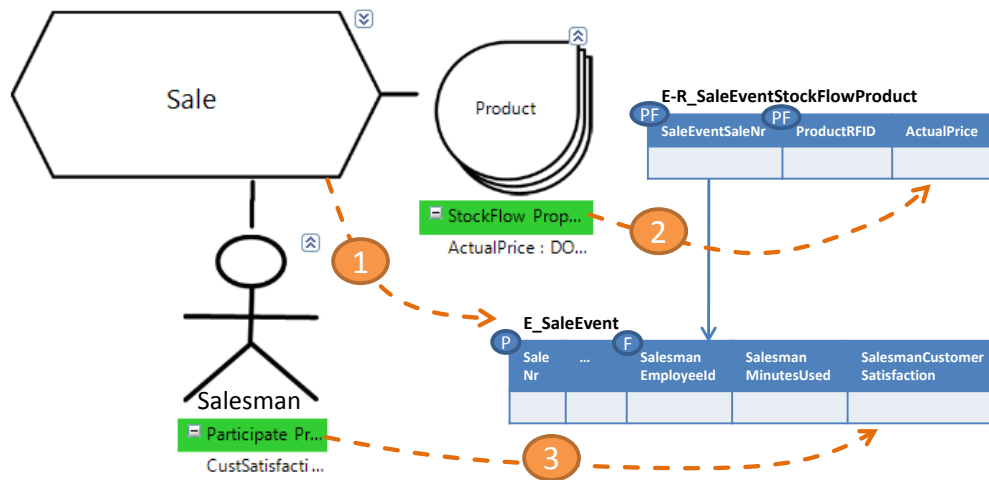


Figure 6.10: Operational Stockflow and Participate Mapping

```

1: for all Stockflow properties do
2:   if Single Resource connected then
3:     add the Stockflow Property as a column with the specified type to the existing event table;
4:   else if Multiple Resource connected then
5:     add the Stockflow Property as a column with the specified type to the existing stockflow table;
6:   end if
7: end for
8: for all Participate properties do
9:   if Single Agent connected then
10:    add the Participate Property as a column with the specified type to the existing event table;
11:  else if Multiple Agent connected then
12:    add the Participate Property as a column with the specified type to the existing participates table;
13:  end if
14: end for

```

Algorithm 6.5: Operational Stockflow and Participation Relationship Mapping Rules

6.2.4 Planning View Mapping

The planning mapping is quite similar to the operational mapping. While the operational view defines what is happening or has happened, the planning view describes what is planned or will happen in the future. For example an order contract (planning view) with the expected agents and resources is made before the actual sale (operational view described above) with the actual agents and resources happens. In our example mapping for the planning view (cf. Figure 6.11) – which we also used in the REA-DSL chapter in Figure 5.27 – a Salesman and a Customer commit in a contract to engage in a Sale event and a Payment event in the future. The exact planning mapping rules are described as pseudo code in Algorithm 6.6.

Contract and Event Type. First, the mapping to the relation model creates (1) a contract table REC_SellingReciprocityContract with a primary key ContractId column for the planning model. Additionally, it contains two columns CommitDecSalesmanId and CommitIncCustomerId for referencing the legally committing inside and outside agent. For the event Sale an event type table ET_SaleEventType (2) is created with the event type

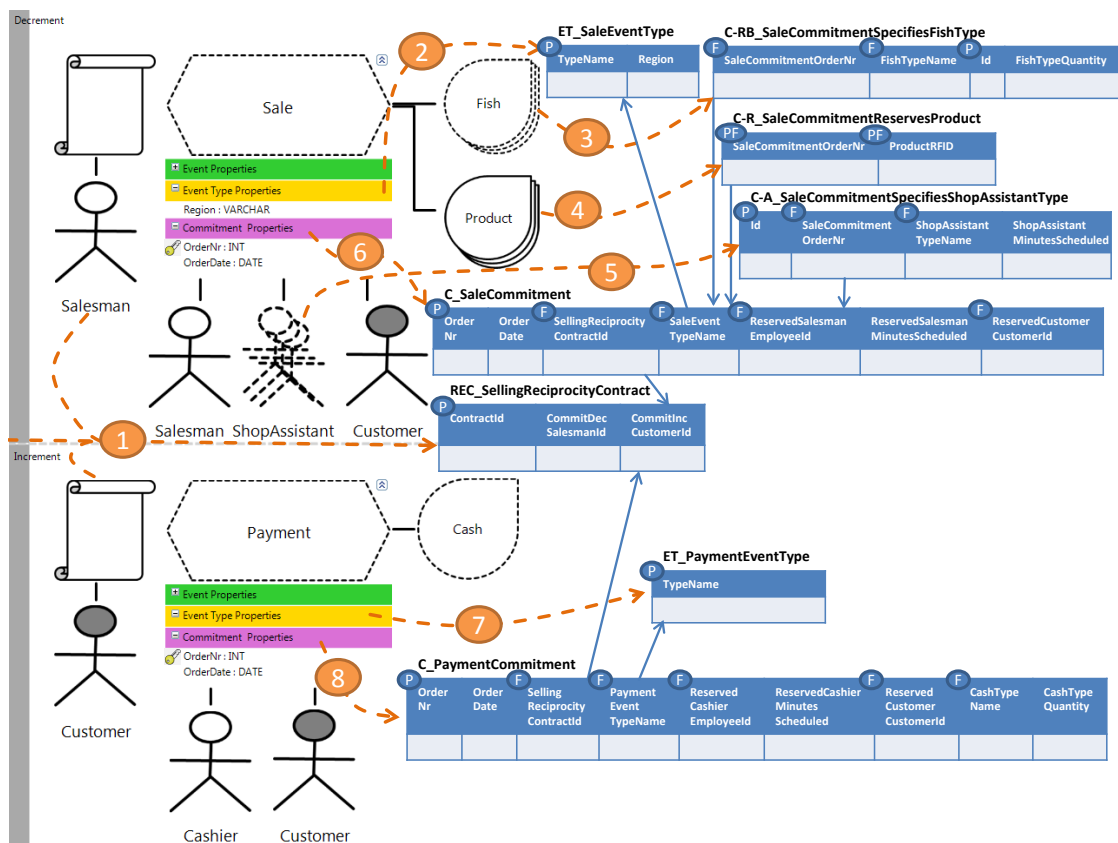


Figure 6.11: Planning Mapping

properties (Region) as columns and a primary key column `TypeName`. Examples for row entries might be "Austrian Sale" or "US Sale". Additionally, for the event `Sale` a commitment table `C_SaleCommitment` (6) is generated with the commitment properties (`OrderNr` and `OrderDate`) as columns and `OrderNr` as primary key. The column `SellingReciprocityContractId` references the contract it belongs to and column `SaleEventType` references the type of the future event.

Agents. Single reserved agents (Salesman and Customer) are referenced directly from the commitment table (`ReservedSalesmanEmployeeId` and `ReservedCustomerCustomerId`). Additionally, for inside agents (Salesman) we also record the minutes scheduled in a separate column (`ReservedSalesmanMinutesScheduled`). Multiple specified agent types (ShopAssistant type) become (5) their own participation table `C-A_SaleCommitmentSpecifiesShopAssistantType`. The table has columns for referencing the `C_SaleCommitment` and the `AT_ShopAssistantType` as well as the minutes scheduled. Because the `ShopAssistantType` might be scheduled multiple times for the same commitment, an additional primary key column `Id` is added.

Resources. Multiple resources (Product) become (4) their own reserve table (`C-R_Sale-`

CommitmentReserveProduct) referencing the corresponding C_SaleCommitment and the individual Product. Multiple bulk resources Fish also become (3) their own specify table (C-RB_SaleCommitmentSpecifiesFishType) referencing the corresponding SaleEvent and the FishType. Different to regular resources, for bulk resources a quantity column is created (FishTypeQuantity). Notice, resources can be marked as being used instead of being exchanged. In such a case, an additional column *minutes scheduled* is generated.

```

1: create Reciprocity Table with the name <PlanningModel.Name>ReciprocityContract/Schedule;
2: - table name either Contract or Schedule according to model type;
3: - add ContractId as primary key column;
4: - add foreign key to the two committing Agents in Agent Table;
5: for all Events do
6:   create an Event Type Table with TypeName of type VARCHAR as primary key;
7:   - add all Type Properties as columns with the specified type;
8:   create a Commitment Table;
9:   - add all Commitment Properties as columns with the specified type;
10:  - make the primary key commitment property the primary key column;
11:  - add foreign key to the Reciprocity Table and the Event Type Table;
12:  for all Connected Agents (specify/reserve participations) do
13:    if Single Agent then
14:      foreign key to Agent Table and add minutes scheduled if is inside agent;
15:    else if Multiple Agent then
16:      create a CommitmentReservesAgents Table;
17:      - foreign key to Commitment Table and Agent Table;
18:      - add minutes scheduled column if is inside agent;
19:    else if Single Agent Type then
20:      foreign key to Agent Type Table and add minutes scheduled if inside agent;
21:    else if Multiple Agent Type then
22:      create a CommitmentSpecifiesAgentTypes Table;
23:      - add Id as primary key column;
24:      - foreign key to Commitment Table and Agent Type Table;
25:      - add minutes scheduled column if is inside agent;
26:    end if
27:  end for
28:  for all Connected Resources (specify/reserve stock-flows) do
29:    if Single Resource then
30:      foreign key to Resource Table and add minutes scheduled if isUsed;
31:    else if Multiple Resource then
32:      create a CommitmentReservesResources Table;
33:      - foreign key to Commitment Table and Resource Table;
34:      - add minutes scheduled column if isUsed;
35:    else if Single Resource Type then
36:      foreign key to Resource Type Table and add minutes scheduled if isUsed;
37:      add quantity (of quantity type) column if is Bulk Resource;
38:    else if Multiple Resource Type then
39:      create a CommitmentSpecifiesResourceTypes Table;
40:      - add Id as primary key column;
41:      - foreign key to Commitment Table and Resource Type Table;
42:      - add quantity (of quantity type) column;
43:      - add minutes scheduled column if isUsed;
44:    end if
45:  end for
46: end for

```

Algorithm 6.6: Planning Mapping Rules

Looking at the bottom decrement entity set of Figure 6.11, the Payment event is mapped to an event type (7) and a commitment table (8) in a similar way as the Sale event was

mapped before. As an additional concept, the single bulk resource Cash is directly added as column CashTypeName in the payment event table referencing the CashType and a CashTypeQuantity column storing the committed amount of Cash being exchanged in the future.

Planning Stockflow and Participation Properties Mapping. As depicted in Figure 6.12 stockflows with resources (types) and participation relationships with agents (types) might also have stockflow properties and participation properties, respectively. The rules applied for the transformation to the relational model are shown in Algorithm 6.7. Looking at our example in the figure the Sale event type has a stockflow relationship to the Product resource series and a participate relationship to the Salesman inside agent. As mentioned in the mapping before, the Sale event type is transformed to a ET_SaleEventType table (1) and a C_SaleCommitment table (2).

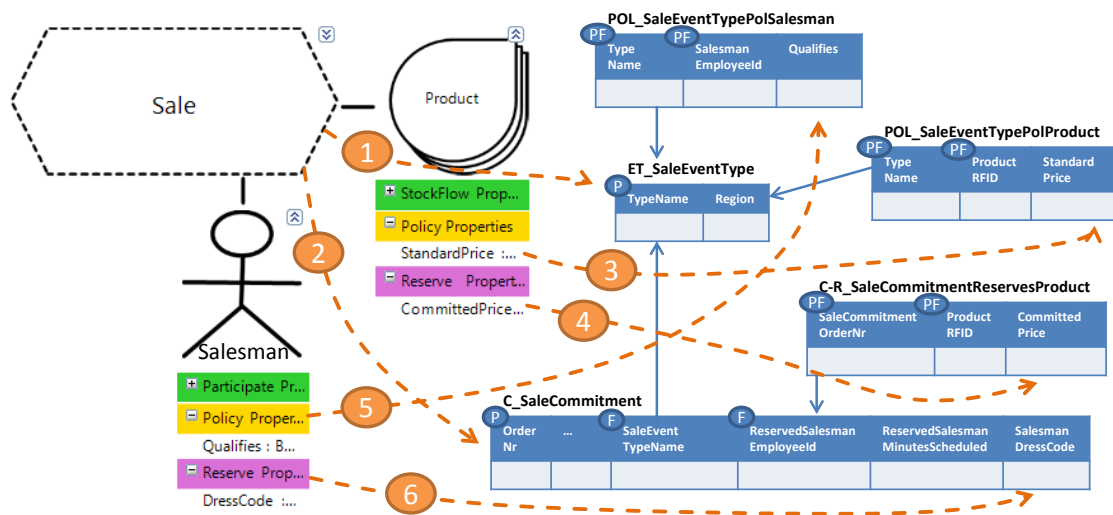


Figure 6.12: Planning Stockflow and Participate Mapping

Stockflow Properties. We first dwell on the mapping of the stockflow relationship, which has policy properties and reserve properties. If policy properties are assigned to the stockflow, a separate table reflecting policies is created. In our example we have a policy property StandardPrice:DOUBLE and thus, we create the table POL_SaleEventTypePolProduct. This table contains all policy properties between the Sale event type and the Product resource. Therefore, also foreign keys (serving together as primary keys) to the Sale event type table and the Product table are included. Additionally, we add the policy property StandardPrice:DOUBLE as a new column StandardPrice with data type DOUBLE to the aforementioned table (3). As for policy tables, a single resource has the same effect as the resource series. If a resource type is connected with the stockflow, the POL_SaleEventTypePolProduct table references the Product Type table instead of the Product table.

The stockflow reserve property `CommittedPrice : DOUBLE` is added as the `Committed-Price` column of the data type `DOUBLE` to the existing `C-R_SaleCommitmentReserves-Product` table (4). If the `Product` resource series would be a single resource, this column would be added to the `C_SaleCommitment` table directly. Furthermore, if the `Product` would be a resource (series) type, the tables would reference the `Product Type` table instead of the `Product` table.

```

1: for all Resource (type) stockflow relationships do
2:   if Policy properties exist then
3:     create an Event Type Policy Resource (Type) Table
4:     - foreign key/primary key to Event Type Table and Resource (Type) Table
5:     for all Policy properties do
6:       - add the stockflow Policy Property as a column with the specified type;
7:     end for
8:   end if
9:   for all Reserve properties do
10:    if Single Resource (Type) connected then
11:      add the stockflow Reserve Property as a column with the specified type to the existing commitment table;
12:    else if Multiple Resource (Types) connected then
13:      add the stockflow Reserve Property as a column with the specified type to the existing
14:      commitment-reserves/specifies-resource(type) table;
15:    end if
16:  end for
17: for all Participate relationships do
18:   if Policy properties exist then
19:     create an Event Type Policy Agent (Type) Table
20:     - foreign key/primary key to Event Type Table and Agent (Type) Table
21:     for all Policy properties do
22:       - add the participate Policy Property as a column with the specified type;
23:     end for
24:   end if
25:   for all Reserve properties do
26:     if Single Agent (Type) connected then
27:       add the participate Reserve Property as a column with the specified type to the existing commitment table;
28:     else if Multiple Agent (Type) connected then
29:       add the participate Reserve Property as a column with the specified type to the existing
30:       commitment-reserve/specifies-agent(type) table;
31:     end if
32:   end for

```

Algorithm 6.7: Planning Stockflow and Participation Relationship Mapping Rules

Participate Properties. Next, we dwell on the mapping of the participation relationship of the planning view, which similar to stockflows has policy properties and reserve properties. If policy properties are assigned to the participate relationship, a separate table reflecting policies is created. In our example we have a policy property `Qualifies : BOOLEAN` and consequently, we create the table `POL_SaleEventTypePolSalesman`. This table contains all the policy properties between the `Sale` event type and the `Salesman` inside agents. Therefore, also foreign keys (serving as well as primary keys) pointing to the `Sale` event type table and the `Salesman` agent table are included. Additionally, we add a policy property `Qualifies : BOOLEAN` as a new column `Qualifies` with data type `BOOLEAN` to the aforementioned table (5). Similar to the policies of resources, an agent series would have the same effect on the policy table as the single agent. If an agent type is connected with the participate re-

lationship, then the `POL_SaleEventTypePolSalesman` table references the `Salesman Type` table (which needs to be created even if it does not have separate type properties) instead of the `Salesman` table.

Last, the reserve property `DressCode : VARCHAR` is added directly to the `C_SaleCommitment` table as the column `DressCode` of the data type `VARCHAR (6)`. If the referred `Salesman` agent is an agent series, this column is added to a newly created table `C-A_SaleCommitment-ReserveSalesman`. Furthermore, if the `Salesman` agent is a `Salesman` agent type, then the table references the `Salesman Type` table instead of the `Salesman` table.

REA-XML

So far, the support for the REA meta-model and the REA-DSL is limited to our tool, which is based on top of Microsoft Visual Studio. However, there are further implementations based on our underlying REA meta-model, e.g., the University of Liechtenstein started the development of the REA-DSL on top of the Oryx [Bus11] business process modeling tool. Even though both implementations are based on the same DSL, the underlying tool specifics prevent an out-of-the-box REA model interchange. In order to come up with a REA description language that is tool independent one may think of using XML Metadata Interchange (XMI) which is designed to describe any meta data whose meta-model can be expressed in MOF. However, XMI showed some deficiencies in practice since there exist a lot of different, incompatible, and vendor-specific XMI dialects and subsets. Furthermore, an XMI-based representation language will require the full support of our meta-model, and will exclude other approaches such as the one by Gailly and Poels [GP07a] who base their conceptual representation on stereotyped UML class diagrams (note these UML models may be expressed in XMI as well, but will be incompatible with our REA meta-model). Others may use ontology editors based on RDF, OWL [GP07b], etc. to describe REA models, which are not compatible with XMI at all. In this case, there is no common ground for a model interchange at all. Accordingly, it seems to be appropriate to provide a serialization of the REA abstract syntax based on an XML schema dedicated to the REA concepts. Of course, we could have used an automatic transformation of our meta-model to an XML schema by the Visual Studio SDK, but this would be a hoax, since it creates a proprietary Microsoft schema including a lot of overhead not acceptable to others. Thus, we developed a dedicated XML schema for REA models, called REA-XML, that allows a precise, textual, tool-independent representation of REA models and that may also serve as an REA model exchange language between different tools. In this chapter we introduce the design of the REA-XML.

Although REA-XML is designed to be tool independent, we started the design process of the REA-XML from our REA meta-model (see Figures 7.3, 7.6 7.12, 7.9, and 7.15). This decision was due to the fact that the REA meta-model is precise in the relationships between the REA core concepts overcoming the vagueness in the original REA ontology. Nevertheless, the resulting XML schema should not be influenced by any DSL-specifics and should provide a

slim and concise tree structure format for REA models. Accordingly, we did not perform a pure UML-to-XML transformation, but aimed for a native XML design respecting all constraints present in the REA meta-model. The design process of the REA-XML is further elaborated in the upcoming sections.

In order to bind a tool specific implementation of REA to the REA-XML, a mapping to the REA-XML schema has to be implemented. In Figure 7.1 we present an overview of such a tool binding for our REA-DSL. On the meta-model layer M2, a mapping from the REA-DSL meta-model to the REA-XML schema, and vice versa, is specified. REA-DSL models on the model layer M1 are mapped to REA-XML models by executing the mapping specification of the M2 layer.

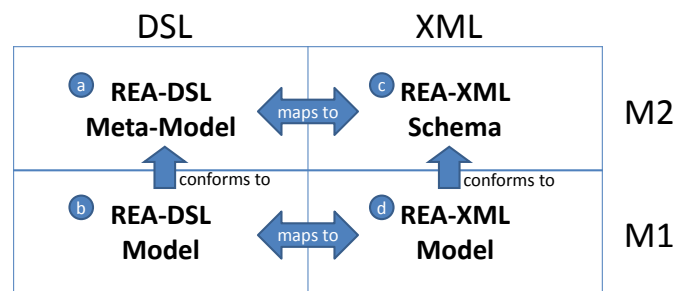


Figure 7.1: REA-DSL to XML Mapping

In this chapter we detail our design decisions when presenting the five REA-DSL views in an equivalent REA-XML schema. In order to help the reader in following our arguments, we have marked equivalent concepts in the REA meta-model (e.g., Figure 7.3) and the REA XSD (e.g., Figure 7.4) with the same markers (white letter in a blue circle). In the XSD, dashed boxes indicate optional elements and stacked boxes indicate that the element can occur multiple times. Furthermore, we decided to present the XSD in a graphical tree notation (used by XML Spy) rather than listing the XML code which is hard to "parse" for humans. The REA-XML consists of five parts shown in Figure 7.2: multiple duality models, multiple planning models, a value chain model, a resource model, and an agent model.

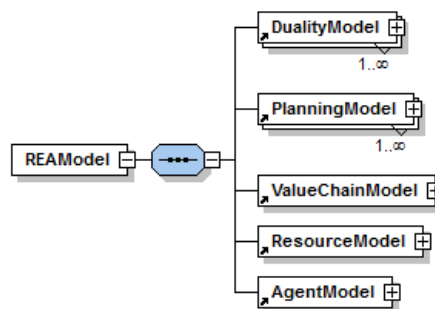


Figure 7.2: REA XSD Parts

First, we present the mapping of the REA building blocks, i.e., the agent view and the resource view. In a next step, the mapping of the dualities in the operational view is described. Afterwards, the mapping of the value chain view which provides an overview of the interconnected dualities of the company is shown. Last, the mapping for the planning view is presented.

7.1 Agent View

The agent view describes all the agents involved in the economic activities. According to the agent view meta-model in Figure 7.3 an Agent (a) can either be an Inside Agent (a1) or an Outside Agent (a2).

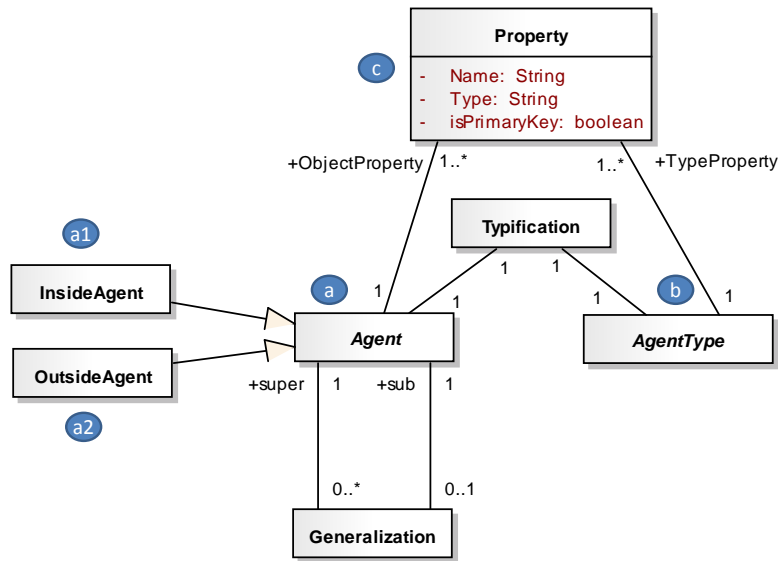


Figure 7.3: Agent Meta-Model

Agents. Consequently, the XSD (cf. Figure 7.4) defines that an Agent Model can contain multiple Outside Agents (a1) and multiple Inside Agents (a2). Additionally, the Agent Model has a name attribute and an id attribute in case it needs to be referenced by an operational view or planning view model. The Agent element consists of three attributes: a name attribute for the agent's name, an id attribute so it can be referenced by the operational and planning view, and a super attribute. The super attribute can reference another (super) agent in a generalization hierarchy according to the Generalization element in the meta-model. Since the Agent Type (b) is in a one to one Typification relationship with the Agent, it is not explicitly covered in the XSD.

Properties. Instead, in the XSD the Agent has a compartment for the Object Properties as well as a compartment for the Type Properties (b) of the AgentType. Properties (c) consist of the name attribute, the type attribute, and the isPrimaryKey attribute. The boolean isPrimaryKey attribute indicates, whether this Property serves as the primary key to uniquely identify an Agent on the instance level or not.

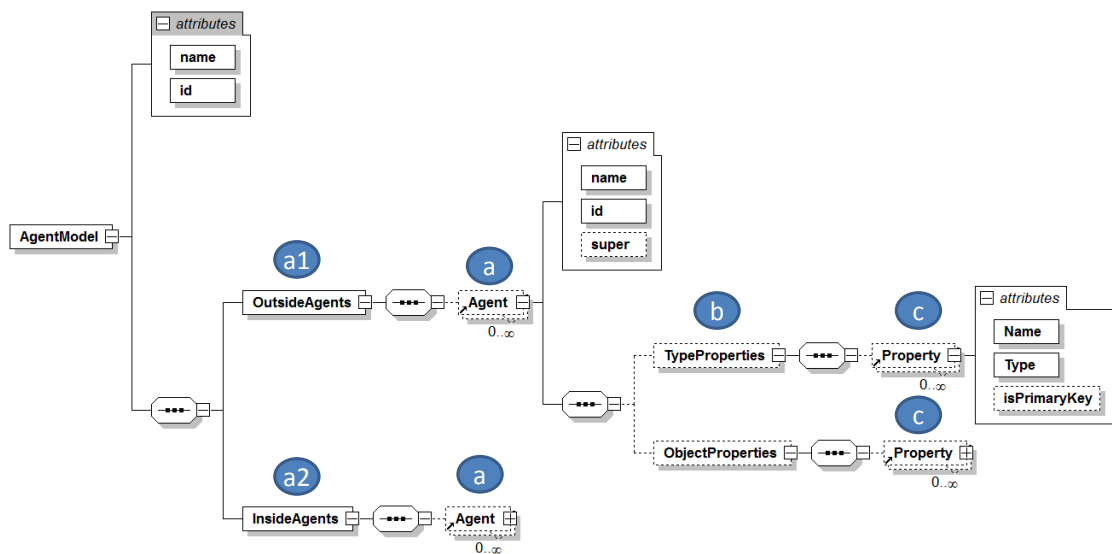


Figure 7.4: Agent XSD

Accompanying Example. Listing 7.1 demonstrates an XML instance of the agent model of the XML schema. This listing is the XML equivalent of the Sy's Fish DSL agent view depicted in Figure 7.5. We have one super agent Employee, which can either be a Shop Assistant, a Cashier, or a Salesman. Employee has various object properties (i.e., Name, Age, EmployeeId) and a type property (i.e., StartingWage). All the sub agents reference the Employee agent by the super attribute and inherit the properties from the super agent Employee. The Salesman agent also specifies an additional type property Region. The outside agent Customer has two object properties Name and CustomerId. In general, the isPrimaryKey attribute in the XML is set for all the properties tagged by a key symbol in the REA-DSL example.

Listing 7.1: Agent XML Model

```

1 <AgentModel name="SysAgents" id="AM001">
2   <OutsideAgents>
3     <Agent name="Customer" id="AG001">
4       <ObjectProperties>
5         <Property name="Name" type="VARCHAR"/>
6         <Property name="CustomerId" type="INT" isPrimaryKey="true"/>
7       </ObjectProperties>
8     </Agent>
9   </OutsideAgents>
10  <InsideAgents>
11    <Agent name="Employee" id="AG002">
12      <TypeProperties>
13        <Property name="StartingWage" type="INT"/>
14      </TypeProperties>
15      <ObjectProperties>
16        <Property name="Name" type="VARCHAR"/>
17        <Property name="Age" type="INT"/>
18        <Property name="EmployeeId" type="INT" isPrimaryKey="true"/>
19      </ObjectProperties>
20    </Agent>
21    <Agent name="ShopAssistant" id="AG003" super="AG002"/>
22    <Agent name="Cashier" id="AG004" super="AG002"/>

```

```

23 <Agent name="Salesman" id="AG005" super="AG002">
24 <TypeProperties>
25 <Property name="Region" type="VARCHAR" />
26 </TypeProperties>
27 </Agent>
28 </InsideAgents>
29 </AgentModel>

```

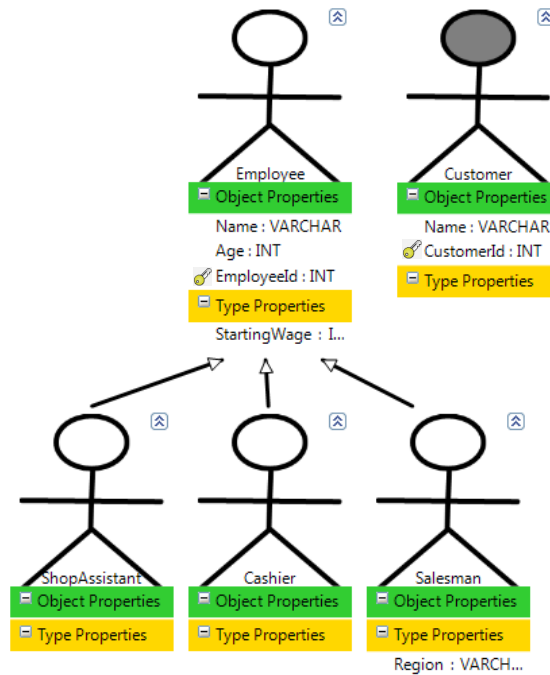


Figure 7.5: Sy's Fish Agent Model

7.2 Resource View

In the resource view all the resources of economic activities which need to be recorded are represented. The meta-model of the resource view (cf. 7.6) seems to look quite similar to the agent view meta-model. However, while Agent Types cannot exist without an Agent, it is possible that a Resource Type (a) exists without a Resource (b) (indicated by the 0..1 to 1 Typification relationship). In this case we speak of a Resource Type as a Bulk Resource. A Bulk Resource can further be specified as a Labor Resource (b1). Consequently, when looking at the agent XSD (cf. 7.7) a Resource Model can have multiple Bulk Resources (b), some of which may be Labor Resources (b1), and multiple Resources (a). Additionally, a Resource Model defines a name attribute and an id attribute in order to enable other models to reference it.

Resources. Resources (a) and Bulk Resources (b) have three common attributes: a name attribute for the resource's name, an id attribute so the operational and planning view can reference the defined Resources/Bulk Resources, and a super attribute. The super

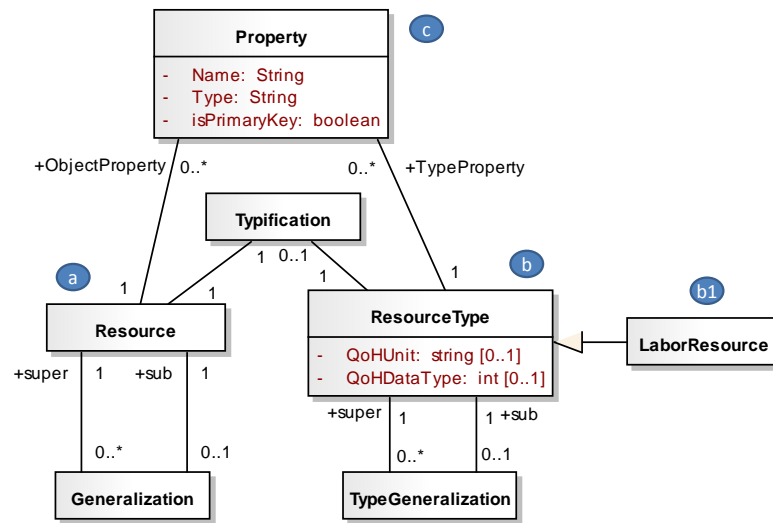


Figure 7.6: Resource Meta-Model

attribute can reference another (super) resource/resource type in a generalization hierarchy according to the `Generalization/Type Generalization` element in the meta-model.

Bulk Resources. A Bulk Resource has three additional attributes. First, the `QoHUnit` attribute describes the unit the quantity (e.g., pounds, kilograms, Euros, Dollars, etc.) of this bulk resource is measured in. Second, the `QoHDataType` attribute defines the data type of this `QoHUnit` (e.g., INT, DOUBLE). Third, the `isLabor` attribute signals, if this Bulk Resource is a Labor Resource or not. By default it is a regular Bulk Resource.

Properties. In the meta-model a Resource (a) can have multiple Object Properties (c) and a Resource Type (Bulk Resource) (b) can have multiple Type Properties (c). Since a Resource (a) needs to be associated to a Resource Type through the `Typification`, a Resource implicitly contains Type Properties. This is also reflected in the XSD. Whereas a Resource (a) has a Type Properties (c) compartment and a Object Properties (c) compartment, a Bulk Resource (b) only has a Type Properties compartment and no Object Properties compartment. Each of these compartments can have multiple Properties (c). Similar to the Property in the agent view, a Property in the resource view has three attributes: a name attribute, a type attribute, and an `isPrimaryKey` attribute. If the `isPrimaryKey` attribute is set to "true", the property becomes (part of) the primary key.

Accompanying Example. Listing 7.2 demonstrates an XML instance of the resource model of the XML schema. This listing is the XML equivalent of the Sy's Fish DSL resource view depicted in Figure 7.8. We specify one labor resource `Labor` depicted by the drop annotated with three small agents. In the XML representation, the `isLabor` attribute is set to "true". `Labor` is recorded in `Minutes` as data type INT. Notice, this QoH information for bulk resources is not depicted in the graphical representation but may be changed through a context menu. A `Product` resource has two object properties `RFID:INT` (which also serves

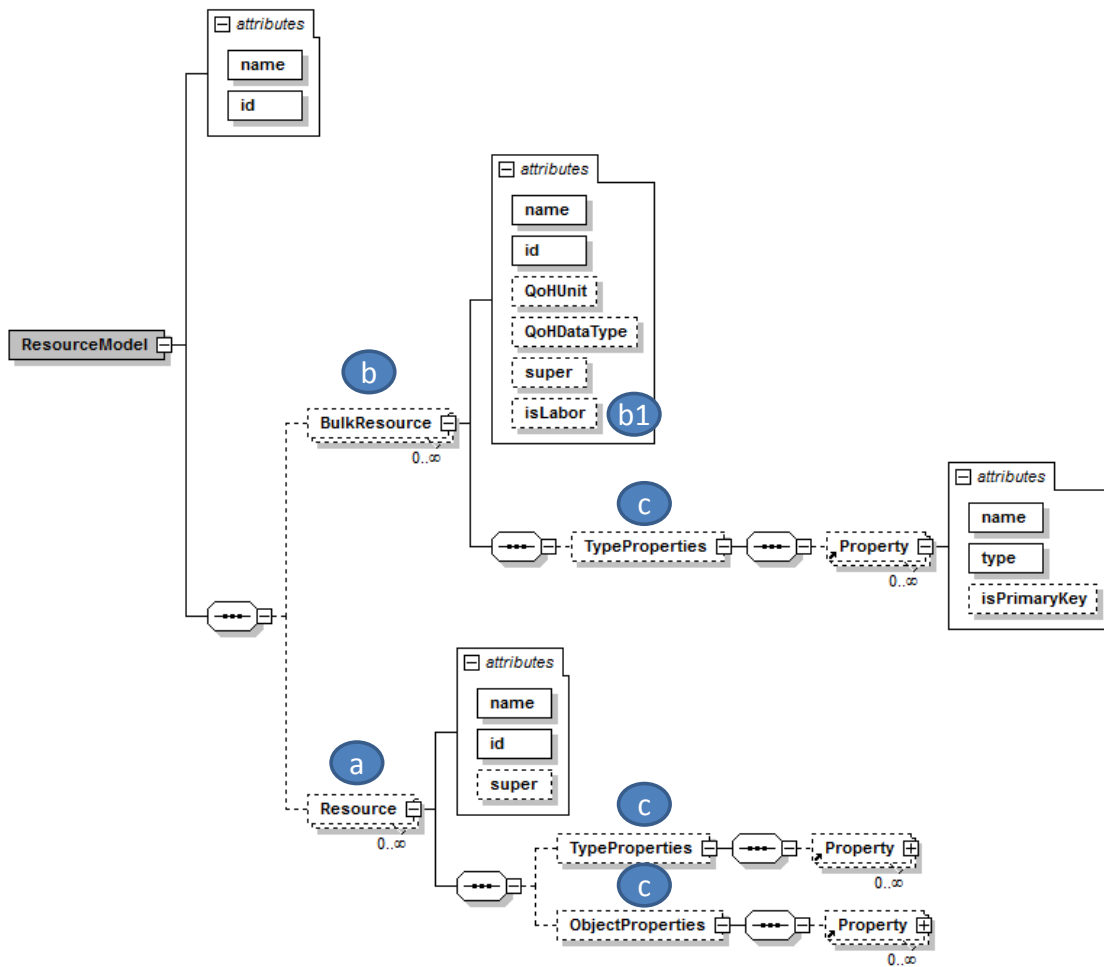


Figure 7.7: Resource XSD

as the primary key) and `Name: VARCHAR` and one type property `Tax: INT`. These properties are also inherited by the two sub resources `Book` and `Fishing Rod`. `Book` additionally has two further object properties `Used: BOOLEAN` and `ISBN: VARCHAR` as well as one further type property `Genre: VARCHAR`. `Fishing Rod` only defines one additional object property `Length: INT`. The bulk resource `Cash` has three type properties: `AccountNr: INT`, `RoutingNr: INT`, and `Currency: VARCHAR`. Notice, since `Cash` is a bulk resource, it cannot contain object properties. Furthermore, the *quantity on hand* for cash is recorded as data type `INT` and as the unit `Euro`. The bulk resource `Fish` contains one type property `StoreTemp: INT`. Its overall amount is recorded in `Pounds` as the data type `DOUBLE`. Last, the regular resource `Truck` has two object properties `RegistrationNr: VARCHAR` (which is the primary key) and `NextService: DATE`. However, it does not define any type properties.

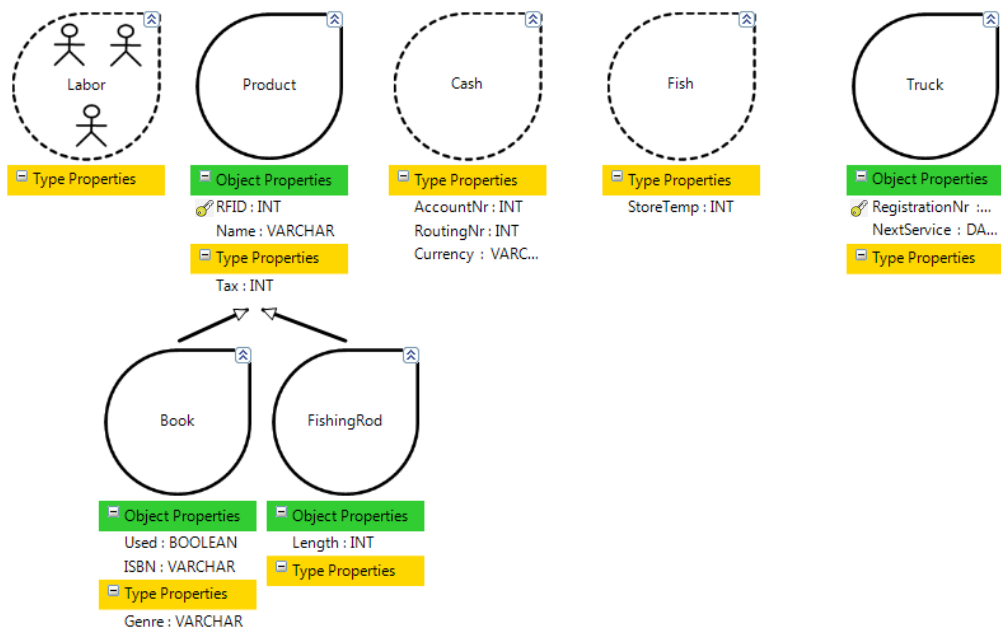


Figure 7.8: Sy's Fish Resource Model

Listing 7.2: Resource XML Model

```

1 <ResourceModel name="SysResources" id="RM001">
2   <BulkResource name="Labor" id="RE001" QoHDataType="INT" QoHUnit="Minutes" isLabor="true" />
3   <BulkResource name="Fish" id="RE002" QoHDataType="INT" QoHUnit="Pounds">
4     <TypeProperties>
5       <Property name="StoreTemp" type="INT" />
6     </TypeProperties>
7   </BulkResource>
8   <BulkResource name="Cash" id="RE003" QoHDataType="DOUBLE" QoHUnit="Euro">
9     <TypeProperties>
10      <Property name="AccountNr" type="INT" />
11      <Property name="RoutingNr" type="INT" />
12      <Property name="Currency" type="VARCHAR" />
13    </TypeProperties>
14  </BulkResource>
15  <Resource name="Truck" id="RE004">
16    <ObjectProperties>
17      <Property name="RegistrationNr" type="VARCHAR" isPrimaryKey="true" />
18      <Property name="NextService" type="DATE" />
19    </ObjectProperties>
20  </Resource>
21  <Resource name="Product" id="RE005">
22    <TypeProperties>
23      <Property name="Tax" type="INT" />
24    </TypeProperties>
25    <ObjectProperties>
26      <Property name="RFID" type="INT" isPrimaryKey="true" />
27      <Property name="Name" type="VARCHAR" />
28    </ObjectProperties>
29  </Resource>
30  <Resource name="Book" id="RE006" super="RE005">
31    <TypeProperties>
32      <Property name="Genre" type="VARCHAR" />
33    </TypeProperties>
34    <ObjectProperties>
35      <Property name="Used" type="BOOLEAN" />
36      <Property name="ISBN" type="VARCHAR" />

```

```

37 </ObjectProperties>
38 </Resource>
39 <Resource name="FishingRod" id="RE007" super="RE005">
40 <ObjectProperties>
41 <Property name="Length" type="INT"/>
42 </ObjectProperties>
43 </Resource>
44 </ResourceModel>

```

7.3 Operational View

The operational view defines the duality of events as well as their exchanged resources and participating agents. Therefore, it also makes use of concepts from the agent view and resource view. These views can be referenced by the multiple duality models of the operational view. The meta-model of the operational view DSL is depicted in Figure 7.9 and its corresponding serialization format is specified by the XSD presented in Figure 7.10.

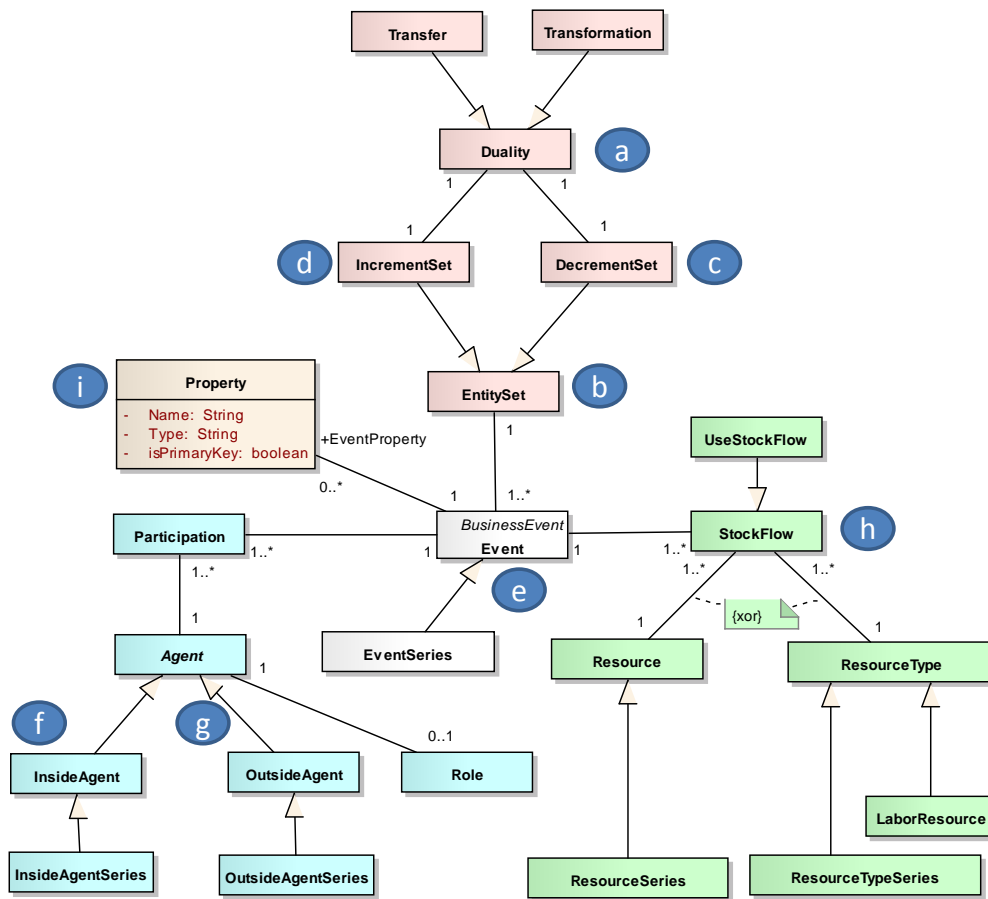


Figure 7.9: REA Duality Meta-Model

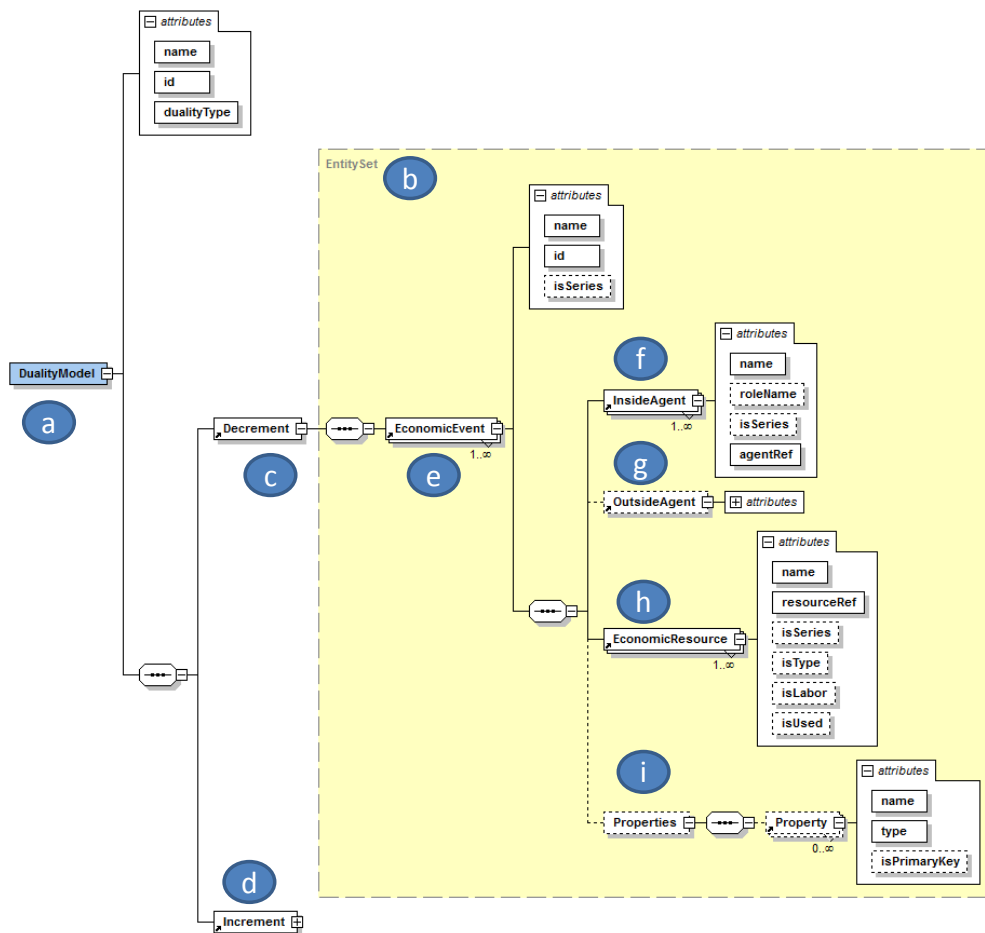


Figure 7.10: Duality XSD

Duality. The concept of a Duality model (a) is mapped to the root element of the duality XSD part. In the XSD, the duality model has a name attribute and an id attribute for referencing purposes. In the meta-model, both Duality (a) and Entity Set (b) have subclasses. The relationship between Duality (a) and Entity Set (b) is modeled explicitly between these subclasses. When taking a detailed look one recognizes that a Duality (a) always covers two Entity Sets (b), one decrementing Entity Set called Increment Set (d) and one incrementing Entity Set called Decrement Set (c). In case of a Transfer the decrementing one is also called Give and the incrementing one is also called Take. In a Transformation the decrement is also called Use/Consume and the increment is also called Produce. In the XSD, we decided that a DualityModel (a) has two child elements: Decrement (c) and Increment (d) representing the two Entity Sets. Both Decrement (c) and Increment (d) are identically structured by inheriting from the complex type Entity Set (b). Furthermore, Duality (a) carries an attribute Duality Type which is an enumeration including the values Transfer and Transformation. Depending

on the instance value of this attribute, one may derive whether the Decrement (c) is a Give or Use/Consume and whether the Increment (d) is a Take or Produce.

Events. In the meta-model, an Entity Set (b) covers one or more Economic Events (e), and each Economic Event (e) resides in exactly one Entity Set (b). Accordingly, an Entity Set (b) in the XSD - no matter whether a Decrement (c) or an Increment (d) - includes one to many child elements representing Economic Events (e). The Economic Event (e) element has a name attribute and a boolean attribute `isSeries`. The latter is per default false, if true it indicates an Economic Event Series. Furthermore, the `id` attribute is used for referencing purposes in the planning view where a commitment has to point to an Event it fulfills.

Participation. By looking at the meta-model, one recognizes that an Economic Event (e) has one to many Participation relations - each connecting exactly one Economic Agent, either an Inside Agent (f) or Outside Agent (g). From OCL constraints not depicted in the meta-model, we know that an Economic Event (e) is connected to at least one but up to many Inside Agents (f) and, additionally, to no (in case of a transformation) or one (in the case of a transfer) Outside Agent (g). It follows that each Economic Event (e) element in the XSD has one to many Inside Agent (f) child elements and zero or one Outside Agent (g) child elements. Inside Agents and Outside Agents have the same attributes. The name attribute should usually be the same name as the referenced agent in the agent view. The `isSeries` attribute specifies, if the agent is an Agent Series or not; the `agentRef` attribute references the corresponding Agent from the agent view. Furthermore a special Role might be applied to the participate relationship by setting the `roleName` attribute of the agent.

Stockflow. Going back to the meta-model, we see that an Economic Event (e) has one to many Stockflow relations - each connecting exactly one Economic Resource (h). Thus, each Economic Event (e) element in the XSD additionally includes one to many Economic Resource (h) child elements. One should note that Economic Resources are re-usable objects in a REA model and, thus, are defined in the previously presented resource view. Accordingly, the child element Economic Resource (h) of the element Economic Event (e) does not define a new resource, rather it includes a `resourceRef` attribute pointing to an economic resource in the resource view. For a better readability in the XML instance, the Economic Resource (h) child element still covers a name attribute. Furthermore, the boolean attribute `isSeries` is used to indicate an Economic Resource Series - this concept is similar to the Economic Event Series mentioned above. Additionally, the `isUsed` attribute specifies, if the resource is just used in the event or if it is consumed/given. This is indicated in the meta-model by a `UseStockFlow` relationship. The attribute `isType` indicates, if the resource is a *bulk resource* (`isType="true"`) or a *regular resource* (`isType="false"`). This attribute has to reflect the type of resource referenced in the resource view. Last, the `isLabor` attribute specifies, if the resource is a Labor Resource. In this case, the `isType` and `isSeries` attributes can be ignored. All the aforementioned boolean attributes (i.e., `isSeries`, `isUsed`, `isType`, `isLabor`) are set *false* by default.

Properties According to the meta-model an Event (e) can have multiple Property (i) elements. Thus, in the XSD an Economic Event contains a Properties (i) compartment.

This compartment can have zero to many `Properties`. A `Property` (i) has to define a name and a `type` attribute. The `isPrimaryKey` boolean attribute indicates, if the `Property` is used as a primary key or not. The default value is *false*. As you may recall from the Chapter 6 "REA-DB", properties can also be defined on stockflows and participate relationships (not depicted in the meta-model and XSD here). These properties are similar to the event properties. The only difference is, that they do not contain primary keys. Therefore, in the XSD you might define `participate` properties for inside agents and outside agents as well as `stockflow` properties for resources in a similar way as for events.

Accompanying Example. Having explained the XML schema of the duality model in detail, we demonstrate a corresponding XML instance in Listing 7.3. Listing 7.3 is the XML equivalent of the DSL Duality Model `Selling` depicted in Figure 7.11. Notice, we just depict one of seven duality models defined by the value chain. The *decrement* part of the XML contains the event `Sale` and the *increment* part contains the event series `Payment` (indicated by the attribute `isSeries` set to "true"). The event `Sale` has an `ID` attribute `EV013`. `Sale` contains two inside agent elements `Salesman` and `ShopAssistant`, both referencing the actual agent in the agent view by the `agentRef` attribute value `AG005` and `AG003`, respectively. Additionally, the `isSeries` attribute for the `Salesman` is set to "true". The outside agent for the `Sale` event is the `Customer` with the `agentRef` `AG001`. Two resources `Fish` (`resourceRef` `RE002`) and `Product` (`resourceRef` `RE005`) are exchanged in the `Sale`. Both have the attribute `isSeries` set to "true". Additionally, `Fish` also has `isType` set to "true" indicating that it is a bulk resource. The `Sale` event also specifies two property elements: `SaleNr` of type `INT` with `isPrimaryKey` set to "true" and `SaleDate` of type `DATE`. The subsequent XML describing the event `Payment` follows the same concepts and is not further explained here.

Listing 7.3: Duality XML Model

```

1 <DualityModel name="Selling" dualityType="Transfer" id="DU007">
2   <Decrement>
3     <EconomicEvent name="Sale" id="EV013">
4       <InsideAgent name="Salesman" agentRef="AG005"/>
5       <InsideAgent name="ShopAssistant" isSeries="true" agentRef="AG003"/>
6       <OutsideAgent name="Customer" agentRef="AG001"/>
7       <EconomicResource name="Fish" isType="true" isSeries="true" resourceRef="RE002"/>
8       <EconomicResource name="Product" isType="false" isSeries="true" resourceRef="RE005"/>
9       <Properties>
10        <Property name="SaleNr" type="INT" isPrimaryKey="true"/>
11        <Property name="SaleDate" type="DATE"/>
12      </Properties>
13    </EconomicEvent>
14  </Decrement>
15  <Increment>
16    <EconomicEvent name="Payment" id="EV014" isSeries="true">
17      <InsideAgent name="Cashier" agentRef="AG004"/>
18      <OutsideAgent name="Customer" agentRef="AG001"/>
19      <EconomicResource name="Cash" isType="true" isSeries="false" resourceRef="RE003"/>
20      <Properties>
21        <Property name="PaymentNr" type="INT" isPrimaryKey="true"/>
22        <Property name="PayDate" type="DATE"/>
23      </Properties>
24    </EconomicEvent>
25  </Increment>
26 </DualityModel>

```

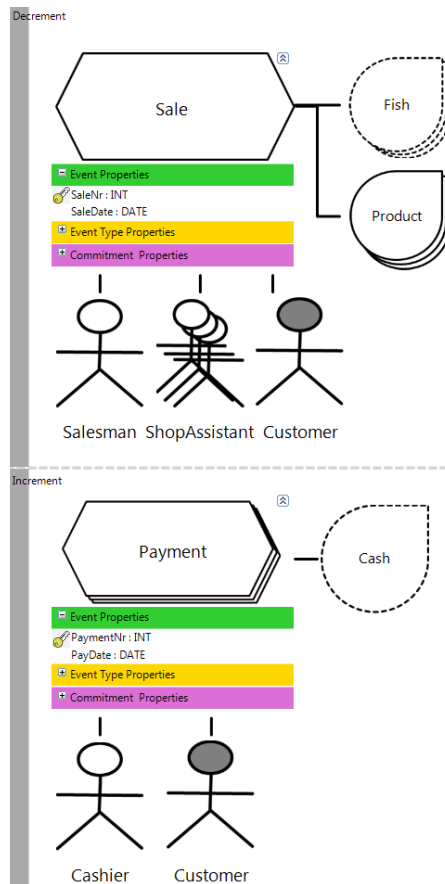


Figure 7.11: Sy's Fish Duality Model

7.4 Value Chain View

Having presented the design decisions for the REA-XML resource, agent, and operational schema part, we now move on to the value chain model part. Again, we use the same markers (single letter in a black circle) in the value chain meta-model (Figure 7.12) and the tree-structure of the value chain XSD (Figure 7.13). The concept of a Value Chain (a) is mapped to the root element of the value chain XSD part. In the XSD, the Value Chain (a) element carries a name attribute and an id attribute for referencing purposes.

Value Activities. In the meta-model, a Value Chain (a) includes one to many Value Activities (b), which can either be a Transfer or a Transformation. Accordingly, the Value Chain (a) element in the XSD includes one to many Value Activities (b) child elements. Each Value Activity (b) child element has a name attribute, an id attribute, and a valueActivityType attribute indicating if the value activity is a Transfer or a Transformation. In the meta-model, a Value Activity (b) may be linked to exactly one underlying Duality model of the operational view. To represent this referencing mechanism in the XSD, each Value Activity (b) element includes a dualityRef at-

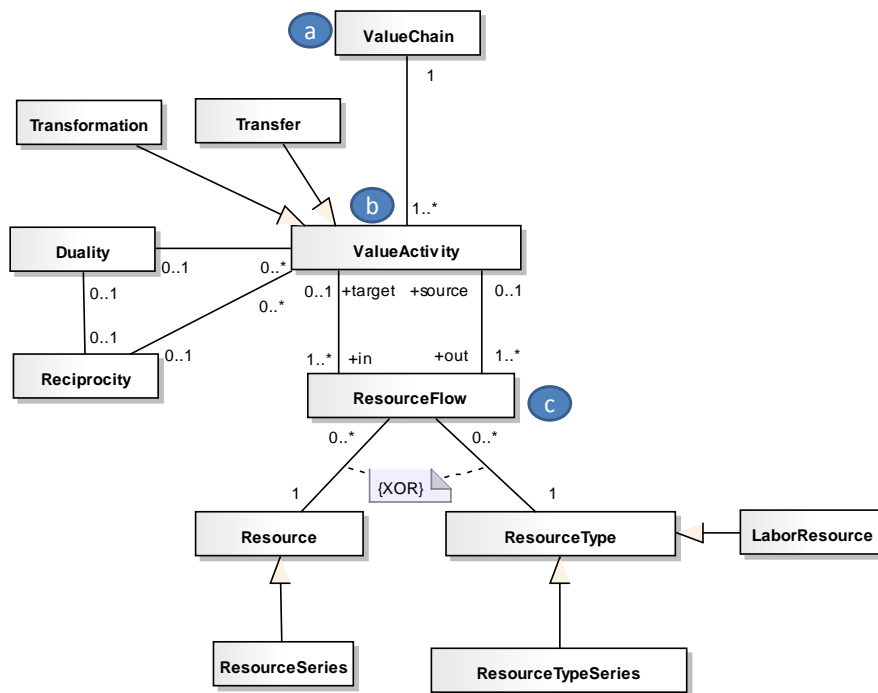


Figure 7.12: Value Chain Meta-Model

tribute to carry an optional reference (IDREF) matching the id of the corresponding duality model. Additionally, the Value Activity (b) may also reference an optional underlying Reciprocity of the planning view. Thus, a Value Activity (b) may include a reciprocityRef attribute to carry a reference (IDREF) matching the id of the corresponding reciprocity model.

Resource Flows. In the meta-model, Value Activities (b) are related to Economic Resource Flows (c). A Value Activity may be the source (out) and the target (in) of many Economic Resource Flows (c). Each Economic Resource Flow (c) starts usually from one Value Activity (source), but may also have no source in the case of partial analyzes of the value chain when a resource is considered as given. Similarly, each Economic Resource Flow (c) leads usually to one Value Activity (target), but may also have no target. In the XSD, we decided not to nest the Economic Resource Flow under the Value Activity, because it usually connects two Value Activities. Instead, an Economic Resource Flow is a direct child element of the Value Chain (a) model. A Value Chain (a) model includes two to many Economic Resource Flows (c), at least one input to the value chain and one output of the value chain. This is in accordance with the "give-take" principle, that for every resource gained a resource needs to be provided. An Economic Resource Flow (c) includes a resourceRef attribute to carry a reference (IDREF) matching the id of the corresponding economic resource of the resource view. In addition to ensure better readability, the resourceName attribute carries the name

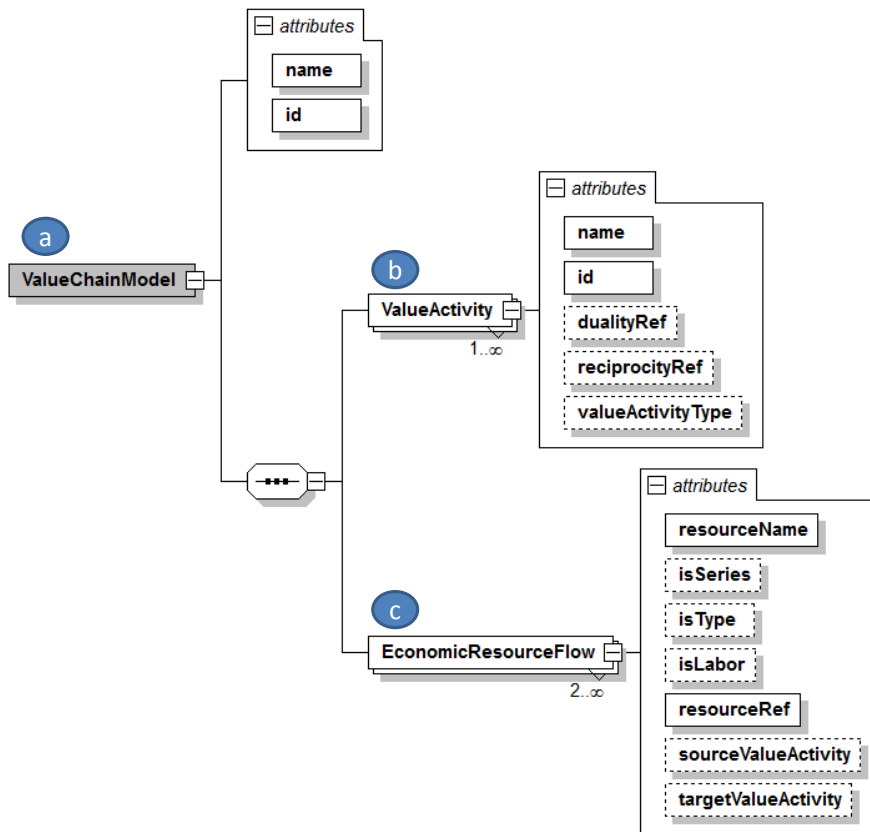


Figure 7.13: Value Chain XSD

of the resource. The three attributes `isSeries`, `isType`, and `isLabor` define the kind of resource (i.e., resource, resource type, resource series, resource type series, or labor resource). By default these attributes are false. Furthermore, the `Economic Resource Flow` (c) element includes a `sourceValueActivity` and a `targetValueActivity` attribute, each carrying a reference (IDREF) matching the id of the source/target `Value Activity`.

Accompanying Example. Listing 7.4 demonstrates an XML instance of the value chain model of the XML schema. This listing is the XML equivalent of the Sy’s Fish DSL value chain depicted in Figure 7.14. It shows a value chain with the following seven value activities: `ProductBuying`, `FishBuying`, `Transport`, `TruckAcquisition`, `Cleaning`, `Selling`, and `Payroll`. For example, the `FishBuying` value activity (line 3 of Listing 7.4) has the `valueActivityType` attribute value `Transfer` and references the underlying duality through the `dualityRef` attribute value `DU002`. The corresponding planning view is referenced by the `reciprocityRef` attribute value `PM002`. The ID of the selling value activity itself is `BP002`. As an example for a resource flow we take the `Fish` resource flow between the value activities `FishBuying` and `Transport` defined on line 15 of Listing 7.4. The `resourceName` is `Fish` and both, `isType` and `isSeries` are set to “true”. The `resourceRef` attribute value `RE002` ref-

references the resource `Fish` with the same ID in the resource view. The `sourceValueActivity` attribute value `BP002` references the `FishBuying` value activity and the `targetValueActivity` attribute value `BP005` references the `Transport` value activity.

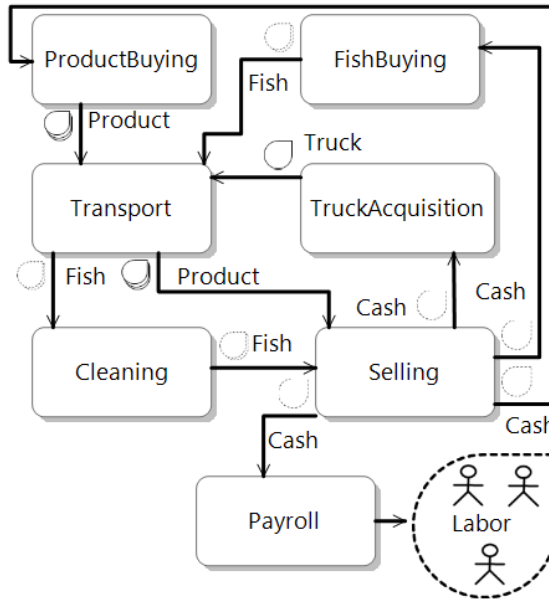


Figure 7.14: Sy's Fish Value Chain Model

Listing 7.4: Value Chain XML Model

```

1 <ValueChainModel name="SysFish" id="VC001">
2   <ValueActivity name="Payroll" valueActivityType="Transformation" dualityRef="DU001"
3     reciprocityRef="PM001" id="BP001"/>
4   <ValueActivity name="FishBuying" valueActivityType="Transfer" dualityRef="DU002" reciprocityRef="
5     PM002" id="BP002"/>
6   <ValueActivity name="ProductBuying" valueActivityType="Transfer" dualityRef="DU003"
7     reciprocityRef="PM003" id="BP003"/>
8   <ValueActivity name="TruckAcquisition" valueActivityType="Transfer" dualityRef="DU004"
9     reciprocityRef="PM004" id="BP004"/>
10  <ValueActivity name="Transport" valueActivityType="Transformation" dualityRef="DU005"
11    reciprocityRef="PM005" id="BP005"/>
12  <ValueActivity name="Cleaning" valueActivityType="Transformation" dualityRef="DU006"
13    reciprocityRef="PM006" id="BP006"/>
14  <ValueActivity name="Selling" valueActivityType="Transfer" dualityRef="DU007" reciprocityRef="
15    PM007" id="BP007"/>
16  <EconomicResourceFlow resourceName="Cash" isType="true" isSeries="false" resourceRef="RE003"
17    sourceValueActivity="BP007" targetValueActivity="BP002"/>
18  <EconomicResourceFlow resourceName="Cash" isType="true" isSeries="false" resourceRef="RE003"
19    sourceValueActivity="BP007" targetValueActivity="BP003"/>
20  <EconomicResourceFlow resourceName="Cash" isType="true" isSeries="false" resourceRef="RE003"
21    sourceValueActivity="BP007" targetValueActivity="BP004"/>
22  <EconomicResourceFlow resourceName="Cash" isType="true" isSeries="false" resourceRef="RE003"
23    sourceValueActivity="BP007" targetValueActivity="BP001"/>
24  <EconomicResourceFlow resourceName="Product" isType="false" isSeries="true" resourceRef="RE005"
25    sourceValueActivity="BP003" targetValueActivity="BP005"/>
26  <EconomicResourceFlow resourceName="Product" isType="false" isSeries="true" resourceRef="RE005"
27    sourceValueActivity="BP005" targetValueActivity="BP007"/>
28  <EconomicResourceFlow resourceName="Fish" isType="true" isSeries="true" resourceRef="RE002"
29    sourceValueActivity="BP002" targetValueActivity="BP005"/>
30  <EconomicResourceFlow resourceName="Fish" isType="true" isSeries="true" resourceRef="RE002"
31    sourceValueActivity="BP005" targetValueActivity="BP006"/>

```

```

17 <EconomicResourceFlow resourceName="Fish" isType="true" isSeries="true" resourceRef="RE002"
    sourceValueActivity="BP006" targetValueActivity="BP007" />
18 <EconomicResourceFlow resourceName="Truck" isType="false" isSeries="false" resourceRef="RE004"
    sourceValueActivity="BP004" targetValueActivity="BP005" />
19 <EconomicResourceFlow resourceName="Labor" isType="true" isSeries="false" isLabor="true"
    resourceRef="RE001" sourceValueActivity="BP001" />
20 </ValueChainModel>

```

7.5 Planning View

In the planning view commitments for economic activities can be defined. The planning view is very similar to the operational view with its dualities. This is due to the fact, that in the planning view agents commit to the future events of the operational view. The planning view contains multiple reciprocities/planning models. In the following, the planning view meta-model is depicted in Figure 7.15. We map all the concepts of the meta-model to a serialization format specified by the planning view XSD (cf. Figure 7.16). Notice, the `ReserveStockFlow` relationship (h) is similar to the `StockFlow` relationship in the operational view and the `ReserveParticipate` relationship is similar to the `Participate` relationship in the operational view. Thus, these concepts have been previously discussed and consequently, we omit further explanations on them in this section. Furthermore, we also omit detailed explanations of other similar concepts and only discuss new attributes.

Reciprocity. The concept of a `Planning` model (a) is mapped to the root element of the planning XSD part. In the meta-model, both `Reciprocity` (a) is either a `Contract` or `Schedule`. Thus, in the XSD, `Reciprocity` (a) carries an attribute `Planning Type` which is an enumeration including the values `Contract` and `Schedule`. If the future events of the planning view are further specified by a dedicated operational view, then the attribute `dualityRef` has to reference the corresponding duality. A `Reciprocity` in the XSD has one `decrement Plan Entity Set` (c) and one `increment Plan Entity Set` (d). A `Plan Entity Set` (b) is similar to an `Entity Set` in the operational view, but additionally also contains a `Commitment (l)` element.

Commitment. The most significant addition of the planning view compared to the operational view is the `Commitment` concept (l) depicted in the meta-model. The commitment specifies, which `Inside Agent` or `Outside Agent` commits to the future `Events`. A `Commitment Series` indicates multiple `Commitments` of the same kind. Therefore, in the XSD a `Plan Entity Set` has to have a `Commitment (l)` element. It has an optional attribute for a name and a boolean attribute `isSeries` indicating a `CommitmentSeries` if set to "true". The `Commitment Agent` refers to the agent who commits to execute the events in the future. It consists of four attributes: the name attribute of the referenced agent with the `id agentRef` attribute, an `isSeries` attribute indicating if it is an agent series, and an `isOutside` attributes specifying an inside or outside agent. Notice, additional commitment attributes are contained directly on the event type.

Event Types. `Event Types` (e) are similar to `Events` in the operational view. One addition is the `eventRef` attribute. It is used for referencing the future `Event` it fulfills. This attribute is only set, if a dedicated operational view refining these `Events` is defined for this planning view.

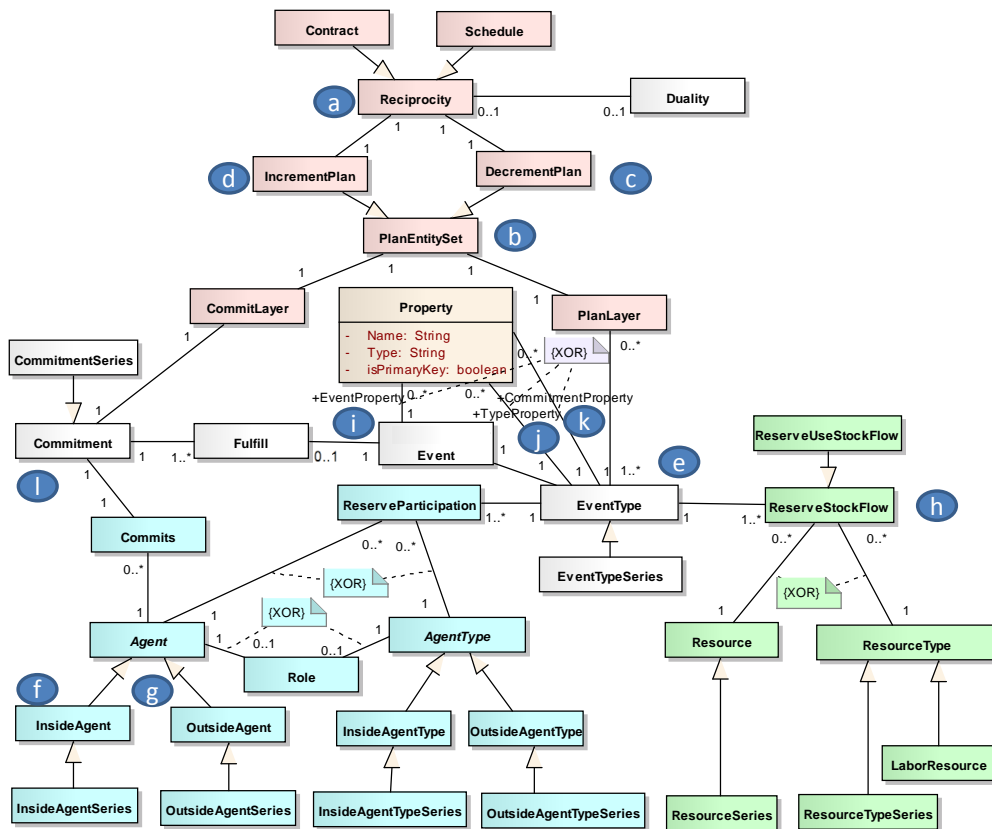


Figure 7.15: REA Planning Meta-Model

Properties. Going back to the meta-model an `Event Type` (e) can have multiple `Type` (j) and `Commitment Property` (k) elements. Thus, the XSD contains a `Type` (j) and a `Commitment Properties` (k) compartment. These compartments can have zero to many `Properties`. Furthermore, if no specific duality model is defined for this reciprocity/planning model, one may model the `Event Properties` (i) connected to the `Event` directly in the planning view. Consequently, in the XSD exists a `Event Properties` (i) compartment. This compartment can as well have zero to many `Properties`.

Similar to the operational view XSD explained before, you might also define properties on stockflows and participate relationships (not depicted in the meta-model and XSD here). `Inside Agent`, `Outside Agent`, and `Resources` may define multiple `Policy Properties` and multiple `Commitment Properties`. `Policy Properties` are used to define attributes for specifying certain policies, standards, and restrictions on the relations between events, agents, and resources. `Commitment Properties` define certain properties for a stockflow or participation relationship in a commitment (e.g., the committed price for a resource in an order).

Accompanying Example. Having explained the XML schema of the planning model in detail, we want to demonstrate a corresponding XML instance in Listing 7.5. Listing 7.5 is the

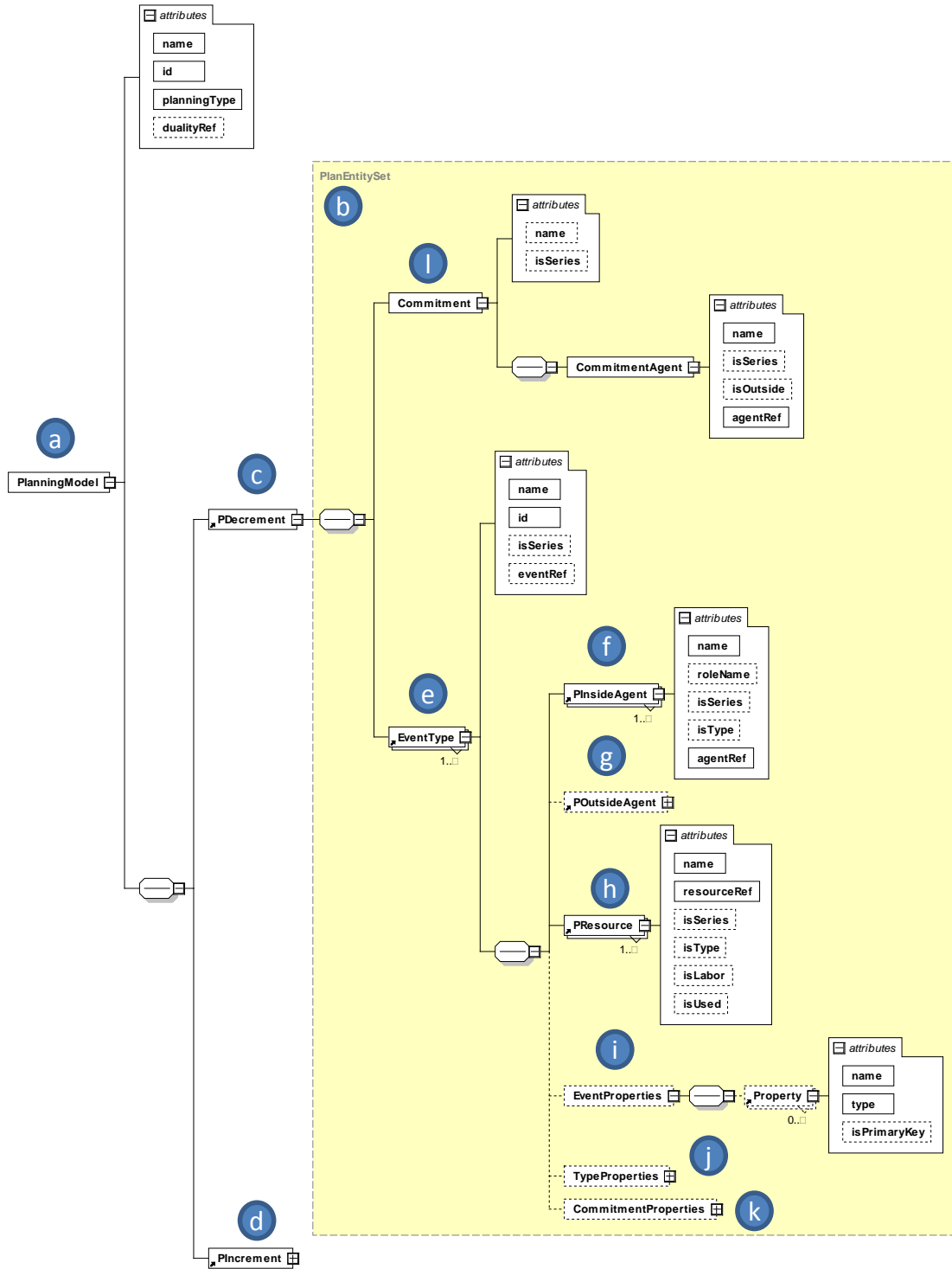


Figure 7.16: Planning XSD

XML equivalent of the DSL Planning Model *Selling* depicted in Figure 7.17 Similar to the operational view, the *Selling* planning view explained here is just one planning view out of seven previously defined by the value chain. In Listing 7.5, there is one event type element *Sale* in the decrement part of the XML and one event type element *Payment* in the increment part of the XML. Since these event types are specified in the XML in a similar manner, we will just dwell on the *Sale* event type here. The *Sale* event type has a unique ID *ET013* and also references the corresponding *Sale* event of the operational view through the *eventRef* attribute *EV013*. Thus, we can specify which event will fulfill this committed event type in the future. The *Sale* event type specifies two inside agents *Salesman* and *ShopAssistant* with the *agentRef* attributes *AG005* and *AG003*, respectively. The attributes *isSeries* and *isType* are set "true" for the *ShopAssistant*. One outside agent *Customer* with the *agentRef* attribute *AG001* is specified. The *Sale* event type specifies various event properties, type properties, and commitment properties. The two event properties are the same as in the previously explained operational view: *SaleNr* of type *INT* (which is also the primary key) and *SaleDate* of type *DATE*. As mentioned earlier, if the planning view and the operational view are conceptually congruent, we do not need to specify a separate operational view and may define the object properties for the dismissed operational view directly in the planning view. We specify one type property *Region* of the type *VARCHAR*. The two commitment properties defined are *OrderNr* of type *INT* (serving as the primary key) and the *OrderDate* of type *DATE*. Last, the *Sale* event type also includes a *Commitment* element. This element contains the agent committing to this future event described above. In our example, this committing agent is the *Salesman* with the *agentRef* *AG005* of the corresponding agent in the agent view.

Listing 7.5: Planning XML Model

```

1 <PlanningModel name=" Selling " planningType=" Contract " id="PM007" dualityRef="DU007">
2   <PDecrement>
3     <EventType name=" Sale " id="ET013" eventRef="EV013">
4       <PInsideAgent name=" Salesman " agentRef="AG005"/>
5       <PInsideAgent name=" ShopAssistant " isSeries="true" isType="true" agentRef="AG003"/>
6       <POutsideAgent name=" Customer " agentRef="AG001"/>
7       <PResource name=" Fish " isType="true" isSeries="true" resourceRef="RE002"/>
8       <PResource name=" Product " isType="false" isSeries="true" resourceRef="RE005"/>
9       <EventProperties>
10        <Property name=" SaleNr " type="INT" isPrimaryKey="true"/>
11        <Property name=" SaleDate " type="DATE"/>
12      </EventProperties>
13      <TypeProperties>
14        <Property name=" Region " type="VARCHAR"/>
15      </TypeProperties>
16      <CommitmentProperties>
17        <Property name=" OrderNr " type="INT" isPrimaryKey="true"/>
18        <Property name=" OrderDate " type="DATE"/>
19      </CommitmentProperties>
20    </EventType>
21    <Commitment>
22      <CommitmentAgent agentRef="AG005" name=" Salesman "></CommitmentAgent>
23    </Commitment>
24  </PDecrement>
25  <PIncrement>
26    <EventType name=" Payment " id="ET014" eventRef="EV014" isSeries="true">
27      <PInsideAgent name=" Cashier " agentRef="AG004"/>
28      <POutsideAgent name=" Customer " agentRef="AG001"/>
29      <PResource name=" Cash " isType="true" isSeries="false" resourceRef="RE003"/>
30      <EventProperties>
31        <Property name=" PaymentNr " type="INT" isPrimaryKey="true"/>
32        <Property name=" PayDate " type="DATE"/>
33      </EventProperties>
34      <CommitmentProperties>

```

```

35     <Property name="OrderNr" type="INT" isPrimaryKey="true" />
36     <Property name="OrderDate" type="DATE" />
37   </CommitmentProperties>
38 </EventType>
39 <Commitment>
40   <CommitmentAgent agentRef="AG001" name="Customer" isOutside="true"></CommitmentAgent>
41 </Commitment>
42 </PIncrement>
43 </PlanningModel>

```

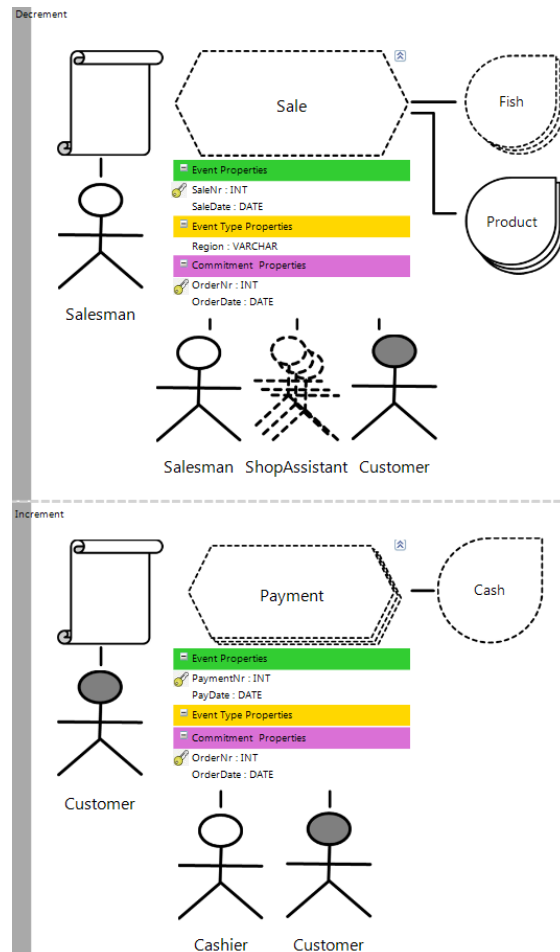


Figure 7.17: Sy's Fish Selling Planning Model

Evaluation and Tool Support

Our evaluation is built upon four pillars: (i) the technical feasibility of the REA-DSL as well as of the mapping to the relational model is demonstrated by a tool implementation, (ii) a functional test based on existing REA models is conducted, (iii) expert interviews were taken, most notable with the founder of REA, Bill McCarthy, and (iv) a usability study with IT and REA professionals was performed. Thus, this chapter is outlined as follows: Section 8.1 dwells on the creation and usage of the REA-DSL tool to demonstrate the (i) technical feasibility. We prove (ii) that existing REA class-like ontology models can be completely redesigned to our REA-DSL in Section 8.2. Next, we conducted (iii) expert interviews [Bog05] to discuss our REA-DSL as well as gain insight into specific REA concept interpretations and sanction our approach by the value modeling and business ontology community in Section 8.3. Last, we perform a (iv) usability study with students and conceptual modeling experts (some with REA knowledge) to evaluate the comprehensiveness of our REA-DSL compared to the class-like REA diagrams.

8.1 REA-DSL Tool

In this section we describe the REA-DSL tool development as well as the REA-DSL tool usage itself. This tool shows the technical feasibility of our REA-DSL. We successfully modeled the whole accompanying Sy's Fish example with our REA-DSL tool and generate the relational model from it according to the mappings provided in this thesis. In Section 8.1.1 we present the environment used to create the REA-DSL. In Section 8.1.2 we give a brief introduction of the structure of the REA-DSL tool and its behavior. However, this is not a thorough user guide and should only highlight the main functions of our tool.

8.1.1 Tool Creation

The graphical DSL used in our solution is based on Microsoft Visual Studio 2010 Visualization and Modeling SDK (VM SDK) [Mic11b] formerly known as Microsoft Visual Studio DSL Tools SDK, which was first introduced in version 2005 of Microsoft Visual Studio. The VM SDK

enables the graphical definition of a DSL directly in Microsoft Visual Studio. Figure 8.1 (previously introduced in Chapter 4) gives an overview of our approach.

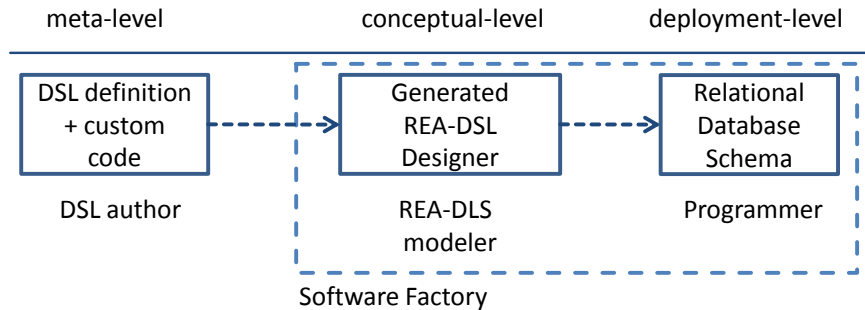


Figure 8.1: DSL Approach

First, a DSL modeler creates the DSL definition together with *custom code* on the meta level. In case of our REA-DSL approach the custom code includes definitions, necessary for the validation of the core component model. In a second step the DSL definition from the meta level is used to generate a REA-DSL designer. Since the REA-DSL designer has been generated from the domain-specific language, it contains only those modeling elements, necessary for the assembly of a REA-DSL model. A REA modeler uses the REA-DSL designer to model REA ontology compliant REA-DSL models as shown in the center of Figure 8.1. Using the built-in template mechanism of the Microsoft VMSDK, the DSL based REA model may be transformed to a relational database schema for the deployment level. Thus, the DSL approach perfectly fits our requirements to first define a REA ontology model on a conceptual level and secondly use the conceptual model to derive relational table schema artifacts for an accounting information system.

Furthermore, Software Factories may leverage the use of DSLs. A Software Factory provides a guidance for the modeler throughout the whole modeling process from creating the model to generating the artifacts. Software Factories may also include smart wizards, which assist the modeler and help to automate repetitive tasks. Wizards may for example be used to derive operational or planning view stubs from the value chain view. Additionally, the validation component allows checking the correctness of a model, helping inexperienced REA modelers to create a correct model. The finished REA-DSL designer can be distributed to REA modelers in an *isolated shell*. The *isolated shell* does not require visual studio and thus, it can be run on a regular Windows PC. The different concepts used for creating the complete tool are explained in the following sections.

8.1.1.1 Specification of the REA-DSL Meta-Model

In the following we briefly introduce the meta-modeling environment for creating a domain-specific language in Visual Studio (in regard to the meta-level in Figure 8.1). The conceptual level was introduced in Chapter 5. The meta-models defined in Chapter 5 have to be remodeled in the DSL Tool in order to adhere to the meta-model facility of the DSL Tool. As seen in

Figure 8.2, the DSL Tool’s meta-modeling canvas consists of two swimlanes: *classes and relationships*, and *diagram elements*. For example, an excerpt of the agent view meta-model shown in Figure 8.2 is referred to as a *domain model* and defines the logical structure of the DSL. A domain model, among other things, consists of *domain classes* (rounded rectangle) and *domain relationships* (rectangles).

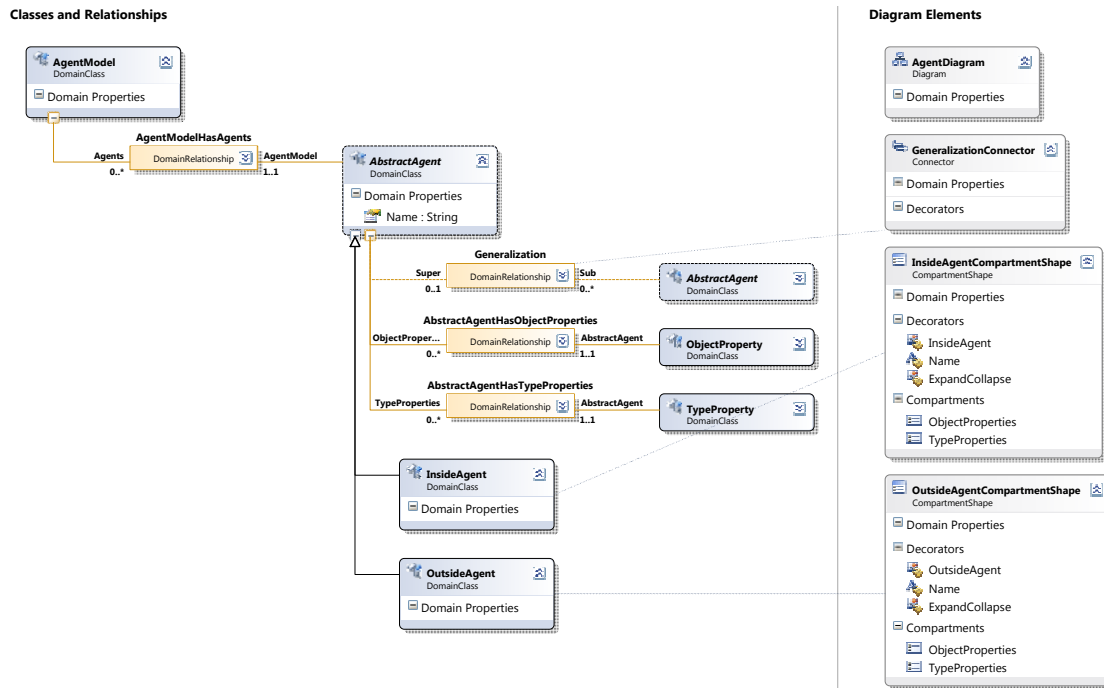


Figure 8.2: Excerpt of the Agent Domain Model

In a nutshell, domain classes define the classes of elements which may occur in an instance of the DSL. Domain classes may have attributes, which are called *domain properties*. Domain properties consist of a name and a data type. Relationships among domain classes are defined by the concept of domain relationships. A domain relationships may either be an *embedding relationship* (shown as a solid line) or a *reference relationship* (shown as a dashed line). Embedding relationships embed the element of the target domain class into the elements of the source domain class. Thus, if the source element is deleted, the embedded target elements are deleted as well. In contrary, reference relationships do not delete referenced elements if the source element is deleted. The embedded references are of particular interest when modeling the operational or the planning view with the DSL. The domain class `AgentModel` on the upper left hand side of Figure 8.2 is the root class of the DSL. There may be at most one root class in a given DSL. In the instance model the root class will represent the parent element for all other classes and their instances. An agent model may contain multiple abstract agents through the agent model has agents embedded relationship. The abstract agent contains a name property. Indicated by the white generalization arrow, an abstract

agent can either be an inside agent or an outside agent. Abstract agents can define a generalization reference to another abstract agent. Furthermore, the abstract agent contains multiple object properties specified by the abstract agent has object properties embedded relationship. It also contains multiple type properties specified by the abstract agent has type properties embedded relationship.

On the right hand side of Figure 8.2 different *diagram elements* of the DSL are shown. Diagram elements are used to define the shape of domain classes and domain relationships in the model instance. They will be generated from the DSL definition. Thus, any shape of choice may be assigned to an arbitrary domain class or domain relationship. Shapes may also be decorated with name (e.g., the name decorator in the InsideAgentCompartmentShape) and icon decorators (e.g., the InsideAgent decorator in the InsideAgentCompartmentShape). A special kind of shape is the compartment shape, which is shown in our example. It enables adding multiple compartments to a shape. This allows modeler to add properties to the model element during runtime. Thereby, a DSL based approach goes beyond classical modeling approaches such as Event Driven Process Chains (EPC) [GHL⁺11] or Unified Modeling Language (UML) [Obj11c], where the modeler is bound to a confined set of elements and their assigned shapes.

We created five different meta-models for the Microsoft DSL Tool corresponding to the REA view meta-models defined in Chapter 5 (i.e., agent view, resource view, operational view, planning view, and value chain view). These five view meta-models are interlinked and can be addressed by the *modeling bus*. The modeling bus allows the definition of *modeling adapters* in order to make models available to other models. Thus, it is possible to reference one model from another and to generate stubs of a target model from the source model. References can either directly reference the whole model (e.g., referencing an operational view) or reference a specific modeling elements within a model (e.g., referencing an agent). It also provides a facility for creating, opening, and saving modeling files.

8.1.1.2 Interaction with the REA-DSL

Custom code enables interacting with the REA-DSL created by the VM SDK. Custom code can either be written in the programming languages .NET Basic or .NET C#. We opted to use C# because it is considered the more advanced and flexible language. Using this language, we write custom code to specify how shapes react if there are actions taken on them. Furthermore, custom code allows modifying the REA-DSL designer, like the appearance and behavior of tools like the property explorer. Additionally, custom code can also be used for defining rules for the DSL. Rules for example can define, that an event in a transfer duality has to have at least one inside agent and at least one outside agent in a REA-DSL. These rules can either be checked right at modeling time or also at the time of saving the model. Consequently, rules can define further constraints additionally to the constraints provided by the underlying meta-model. If any of these rules are broken, an exception will be shown in a specific tool bar for warnings and errors.

8.1.1.3 REA-DSL to Relational Model Mapping Implementation

In Chapter 6 "REA-DB" we have introduced the mapping between the REA-DSL and the Relational Model. This mapping is described as pseudo code. Model-to-code transformations can be used to implement this mapping in our REA-DSL designer. One powerful model-to-code transformation tool is the text template transformation toolkit (T4) [Mic11a]. It is a template-based code generation engine developed by Microsoft. The T4 templates are defined by processing directives, text blocks, and code blocks and have the file extension ".tt". Code blocks may be written in C# or Visual Basic .NET. In the REA-DSL designer we used the T4 text templates to derive the relational models in terms of an SQL script from the REA-DSL models. These scripts can be run by the REA modeler to generate the according artifacts. Afterwards, the generated SQL statements can be loaded into a relational data modeling tool such as MySQL Workbench [Ora11] to get a graphical representation of the relational model. An excerpt of the T4 text templates for agents (1), its SQL statements output (2) and a relational graphical representation of the MySQL Workbench (3) is shown in Figure 8.3.

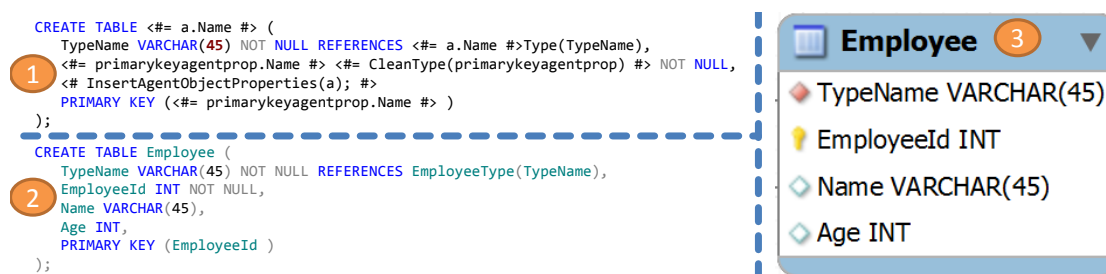


Figure 8.3: T4 Template to SQL Statements to Relational Model

8.1.2 Tool Description

The REA-DSL designer provides following two main features: (i) the creation of a REA-DSL model and (ii) the generation of relational database schema based on SQL statements. In Section 8.1.2.1 we describe the different existing parts of the REA-DSL tool. After that, we explain the general procedure to generate a REA-DSL model in Section 8.1.2.2.

8.1.2.1 Tool User Interface

Figure 8.4 shows the REA-DSL designer with an open `selling planing` view model file in the center. The user interface of the REA-DSL designer consists of five main parts: (1) the canvas, (2) the toolbox, (3) the error list, (4) the property window, and (5) the solution explorer. Additionally, a wizard supports the modeler in creating the models (not depicted in the figure).

(1) Canvas. Most important, the canvas in the center is the hearth of the application. It is the modeling area for the REA models. The modeler can move elements around and design his REA-DSL model. If an element which has dependent elements is deleted – for example event

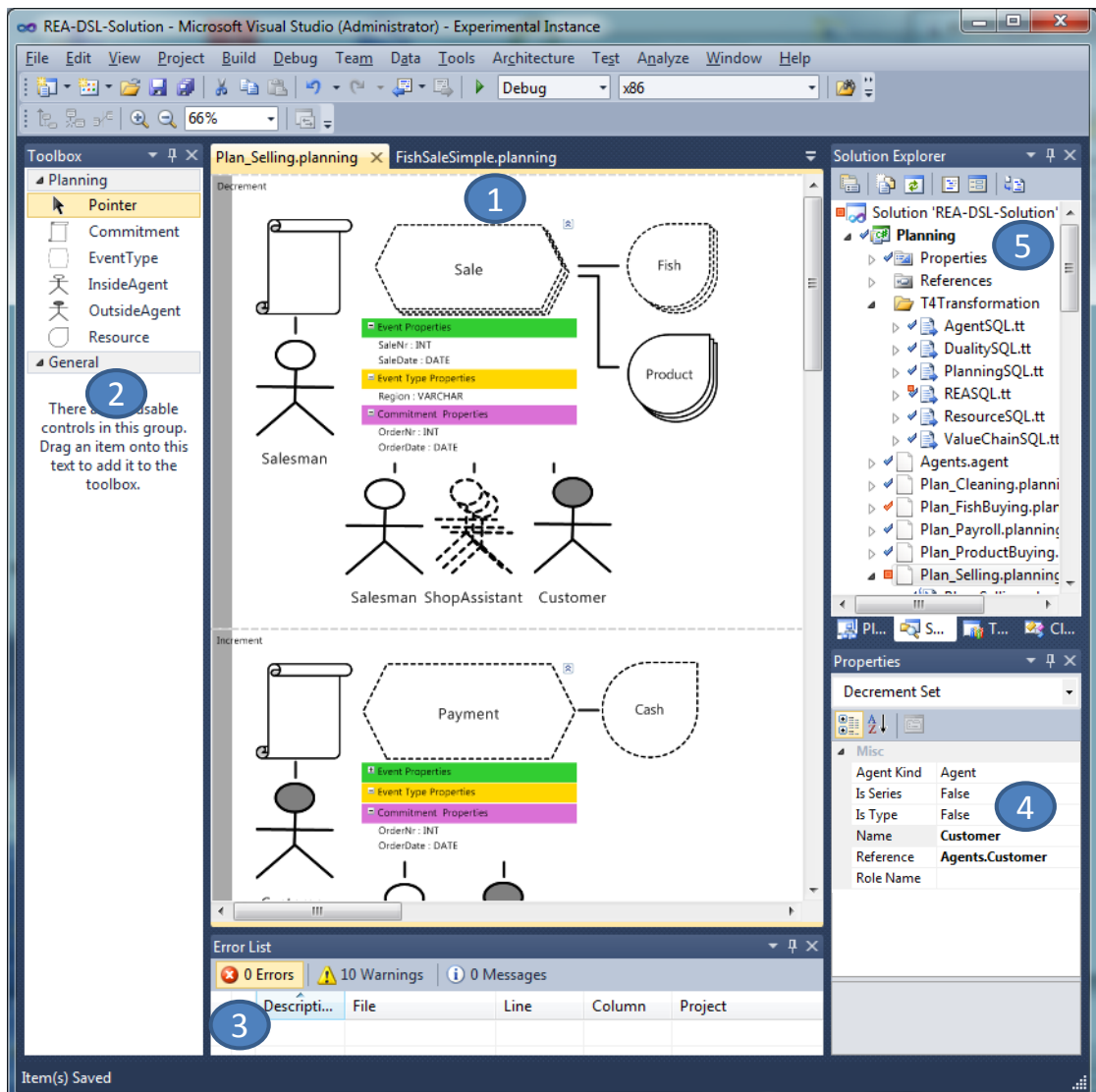


Figure 8.4: REA-DSL Tool

type `sale` and agent `salesman` – the dependent element is deleted as well. Copy and paste facilities help the user to save time with repeating tasks. Elements with properties have a little double arrow on their top right corner (cf. `sale` event type). By clicking this arrow, the user can collapse and expand the property compartments. There can be multiple modeling files open at the same time. An easy to use tab interface enables switching between the open modeling files.

(2) Toolbox. On the left side of the designer you can see the toolbox with the different modeling elements available for the current view. In Figure 8.4 we show a planning view on the canvas,

and thus, only elements (toolbox items) required for the planning view are shown in the toolbox (i.e., commitment, event type, inside agent, outside agent, and resource). A toolbox item can be dragged and dropped on the canvas. During this action, the tool checks if the combination of the elements is valid. For example, if you try to drop an inside or outside agent on a resource, the tool will not allow you to do so. This is due to the fact, that the meta-model behind this REA-DSL planning view model does not define such a relationship. If you try to drop an inside or outside agent on the event type symbol, the tool will put the agent on the canvas and connect it to the event type you dropped it on. In this case, the meta-model defines such a relationship (stock-flow) and the tool allows this action.

(3) Error list. At the bottom of the tool is the `error list`. This error list will show violations of the REA ontology. As already mentioned before, an event of a transaction duality always has to have at least one inside agent and one outside agent. If this rule is violated and the user tries to save his model, an error pop-up will occur and a detailed message saying that "there has to be at least one inside and one outside agent in an event of a transaction duality" will appear in the error list.

(4) Property window. The bottom right of the REA-DSL designer shows the property window. This window always shows the various properties of the selected element of the canvas. Thus, when an inside or outside agent is selected, different properties will be shown according to the selected modeling element. Properties for a resource are for example its name, its reference, its type, or a series flag. When changing the type or the series property, the notation of the resource will change accordingly. If the type property is set to true, the resource will be shown as a dashed drop, if it is set to false, the resource will be shown as a solid drop. If series is set to true, a stack of drops is shown. The reference stores the location of the corresponding resource element in the resource view. If the user clicks on the name property, he is provided with a list of all possible resource names generated from the resources in the resource view.

(5) Solution explorer. The top right of the REA-DSL designer contains the solution explorer. It provides an overview of all the modeling files and the text templates for generating the SQL scripts. Multiple projects can be loaded in the solution, each specifying a different REA business model. One project consists of multiple interlinked files: one agent file, one resource file, one value chain file, multiple planning files, and multiple duality files. By double clicking on these files they get opened in the canvas and can be modified. The text transformation templates for creating the SQL scripts are stored under the `T4 Transformation` folder. To generate the SQL scripts, the user has to right click on the `REASQL.tt` file and choose `run custom tool`. A new file called `REASQL.sql` appears below it containing all the SQL statements required for creating the database structure. The user can now use a third party relational table modeling tool (e.g., MySQL Workbench) to import the `REASQL.sql`. This provides the user with a graphical representation of the relational tables created from the REA-DSL. Last, the user can decide to change the solution explorer to the planning view explorer. The planning view explorer provides a tree view on the model in the canvas.

Wizards. The tool provides wizards supporting the user in creating REA-DSL models. Wizards are usually dialog boxes which guide the user through various steps to perform a certain task. Thereby, the user usually saves time and the complexity of the task is reduced. Lets assume that we have a value chain and want to create an operational view for one of the value activities. Without the wizard, the user has to create a new operational view from scratch, defining all the events, resources, and agents by hand. Furthermore, he has to make sure that the resources used are in conformance with the resource flows of the value activity. Additionally, he also has to set the duality reference of the value activity to the newly created operational view. Evidently, there are many manual steps involved. With the wizard, the user just needs to double click on the value activity. The wizard asks, if an operation view should be created for this value activity. If the user confirms, an operational view stub, in conformance to the value activity and its resources, is automatically created in the background. The user just has to modify the involved agents of the generated operational view and is done.

8.1.2.2 REA-DSL Modeling Steps

This section provides directions to model a REA-DSL model. Notice, it is not required to stick to this sequence, you can also switch between views and alternately add modeling elements in different views. After running the REA-DSL designer the user has to create a modeling solution with a new REA-DSL project. A clean project consists of three empty files: the agent view, the resource view, and the value chain view. In Figure 8.5 seven steps creating a REA-DSL and transforming it to a relational model are depicted. These steps are explained in detail in the following:

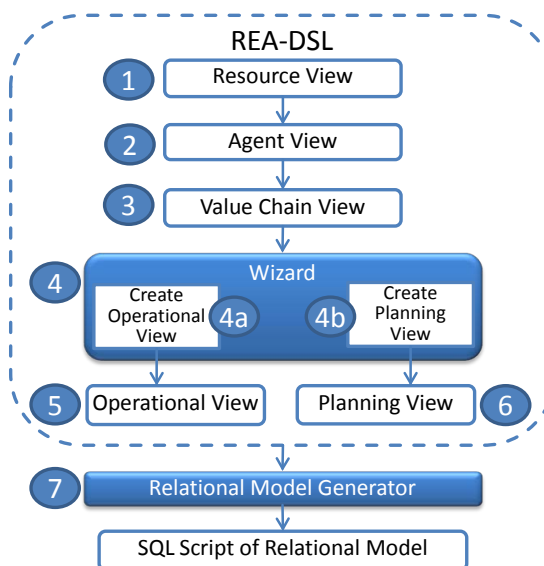
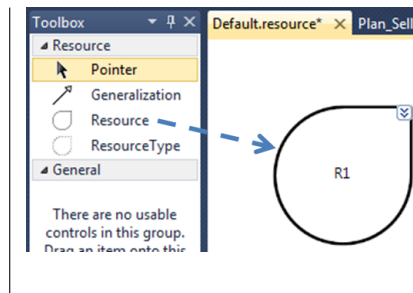


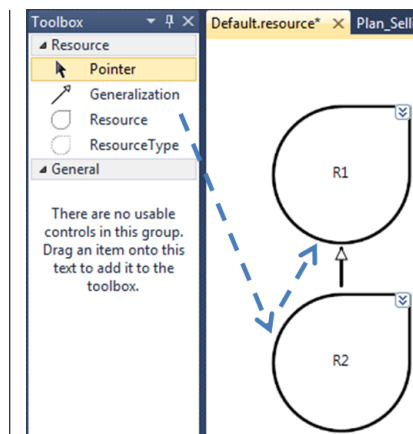
Figure 8.5: REA-DSL Modeling Steps

(1) Defining resources.

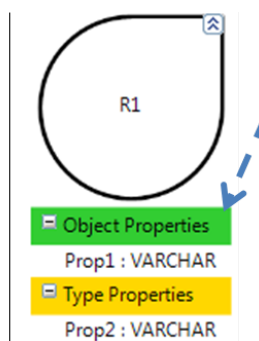
First, you can define all the resources in your company in the resource view. Therefore, you drag a regular or bulk resource (resource type) from the toolbox and drop it on the canvas. A standard name for the resource is provided. It can be changed by double clicking on the name and providing a new one.



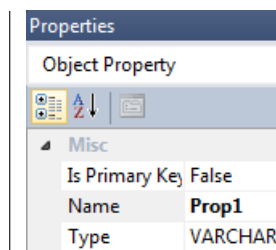
You may define a generalization between the more specific resource and the more general resource. In order to do this, you need to select the generalization arrow in the toolbox and click first on the more specific resource and then on the more general resource. Generalizations can only be between regular resources or between bulk resources, respectively. You are not allowed to mix resource types in a generalization hierarchy.



In a further step you can define the object properties as well as the type properties. If the property compartments of the resource are not shown, you can display them by clicking on the small double arrow on the top right of the resource. To add a property, you need to right click on the desired compartment and choose "add new property".

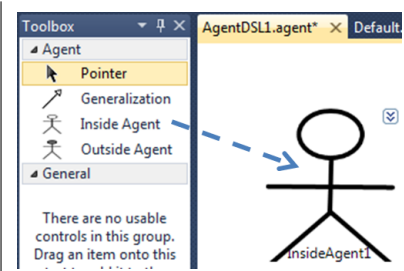


Specifics of the property can be changed in the property window. Make sure you define the primary key for each resource. More specialized resources inherit the primary key from the more general resource.



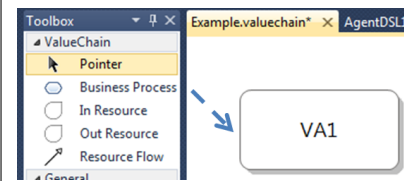
(2) Defining agents.

Next, you define all agents inside and outside of your company in the agent view by dragging the stick figure from the toolbox and dropping it on the canvas and renaming it. You can also generalize your agents by adding a super inside agent employee or add properties to them. This works in the same way as previously described for the resources. Inside and outside agents cannot be mixed in a generalization hierarchy.

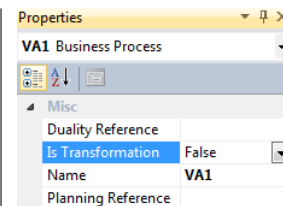


(3) Defining the value chain.

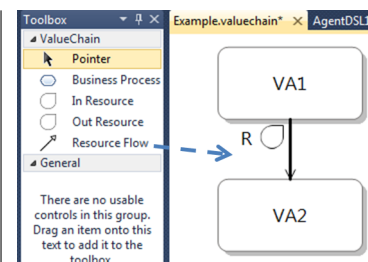
After you have defined all the REA building blocks you can go on to define the economic phenomena your company bases its business on in the value chain. You need to specify all value activities such as selling, buying, producing, etc. Therefore you put a value activity (business process) from the toolbox on the canvas and rename it.



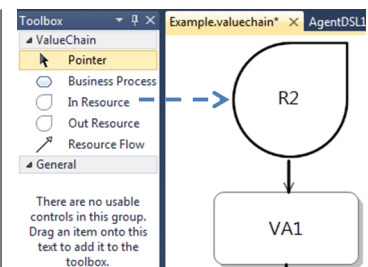
In the property window, you can define if a value activity is a transfer or a transformation. This will also affect the generation of the operational view and the planning view described in a later step.



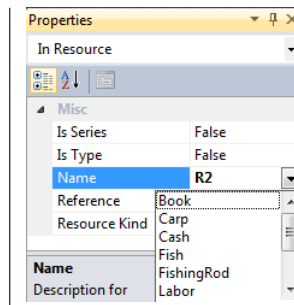
Next, you connect the value activities with each other by resource flows taken from the toolbox. Drag it from the source value activity to the target value activity. You can define multiple resource flows between the same source and target value activity.



If you want to add a resource flow which does not have a source or target value activity, you use an *in resource* or *out resource* element from the toolbox and drop it on the value activity.

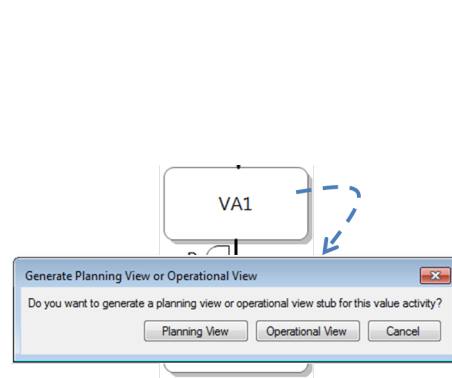


In order to set the name of the resource to a name from the resources you defined in the resource view, you first have to click on the resource flow. Any of the available resource names can then be selected from the name drop down box in the property window. Additionally, in the property window you can define the type of the resource and whether it is a series or not.



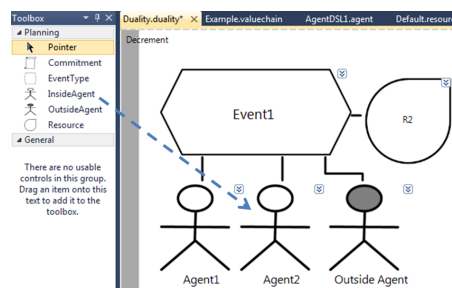
(4) Using the Wizard to create operational and planning views.

Next, you need to define the events or commitments of your REA-DSL in operational views or planning views, respectively. You can either do this by hand from scratch or by using the provided wizard. The wizard saves you time by generating operational view and planning view stubs which conform to the value activities defined previously. Only minor modifications have to be applied on the models afterwards. Thus, we explain how to create the operational view and planning view for a specific value activity using the wizard. Therefore, you double click on the desired value activity in the value chain view. The wizard will open and ask you, if you want to (4a) create the operational view or if you want to (4b) create the planning view for this value activity.

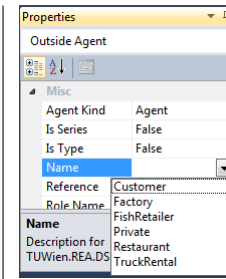


(5) Defining the operational view.

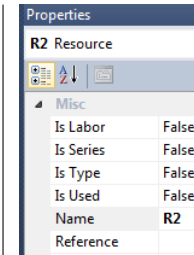
If you chose (4a) to generate the operational view, a stub for the operational view with one incrementing and one decrementing event is generated and includes all resources from the related value activity in the value chain view. Additionally, in the case of a transfer value activity, one inside and one outside agent is added to each event. In the case of a transformation, one inside agent is added to both events. You can also add additional agents to events.



Next, you need to rename the agents by selecting one of the provided names from the name property drop down box in the properties view. This drop down box contains all names of the agents defined in the agent view. Additionally, you can also define if the agent specifies a series by setting the *is Series* property to true.



For each resource, you may define in the property window if it is consumed or if it is just used in the event. If an event, resource, or agent can occur multiple times, you need to change the value of *is Series* in the property window accordingly.

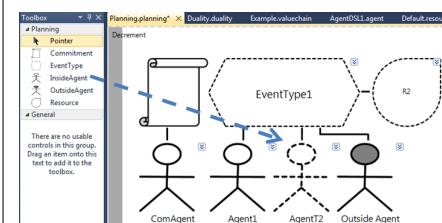


Additionally, you have to define the event, stockflow, and participate properties in the green compartments and specify a primary key. If you do not see the property compartments, click on the little double arrow on the top right of each element. If the property is used as (part of) the primary key can be set in the property window.

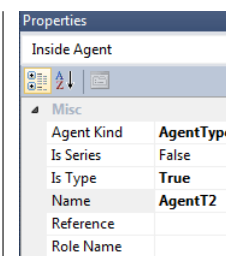


(6) Defining the planning view.

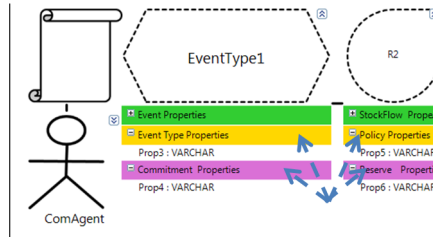
If you chose (4b) to generate the planning view, a stub for the planning view with one incrementing/decrementing event type as well as one incrementing/decrementing commitment scroll with a committing agent gets generated. Additionally, the resources and the remaining agents get generated in the same way as described in the operational view. You can rename and add agents similar to the operational view.



In order to use the type of a resource or agent, the value of *is Type* in the property window has to be true. If an event type, resource (type), or agent (type) can occur multiple times, you need to change the value of *is Series* in the property window accordingly.

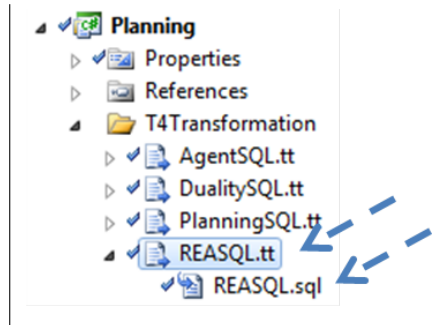


You need to define for event types, stockflow relationships, and participate relationships the commitment properties in the purple compartment and the type properties in the yellow compartment. If the property is used as (part of) the primary key can be set in the property window.



(7) Generating the relational schema.

Once you are done, you can create the relational schema in terms of an SQL script. To do so, right click on the REASQL.tt file in the T4Transformation folder and choose "Run Custom Tool". This generates a file called REASQL.sql which contains all the SQL statements to create a relational database schema. This file can be imported by a database modeling tool of your choice (e.g., MySQL Workbench) or it can be used directly by the database system to generate the relational tables.



8.2 Redesigning Existing REA Models

To evaluate the completeness of our REA-DSL, we conducted a functional test by redesigning existing REA models specified by class diagrams to our REA-DSL. We prove, that we can specify all REA concepts from the original class models in our REA-DSL. Bill McCarthy provided us with the twelve REA class models he uses in his classes. These REA models are actually based on real companies in the United States. These twelve REA class models differ in the structure among each other concentrating on different REA concepts in order to demonstrate various aspects of the REA ontology. Thus, we are confident of having models considering all REA concepts for this evaluation. Searching through the Internet and literature, we came up with more examples. However, these examples are often small and simple and their concepts are already covered by the more sophisticated examples we received by Bill McCarthy. This is not astonishing, since REA defines a quite restrictive way to describe business phenomenas and thus, even though contextually different, many REA models consist of the same structure. Thus, we stay with the twelve examples provided by Bill McCarthy for this functional test. The examples and their characteristics are listed in the following and raise with their complexity from one to the other:

- **University Slum Lords:** Acquisition of resources; identifiable inventory
- **Boston Bottle:** Acquisition and revenue cycle; bulk inventory
- **Marilyn Monroe Makeovers:** Bulk inventory and services; some typing
- **Native Alaskan Aircraft Expeditions:** Revenue cycle; typed employees

- **Vivian’s Fashion Factory I:** Acquisition cycle for nail inventory; complex commitments
- **Brian’s Gemini Music:** Revenue cycle; many typifications
- **Flint X-Ray:** Two phase revenue cycle; agents for commitments
- **South Shore Petroleum:** Multiple transfers; multiple commitments
- **Western Michigan Office Furniture:** Conversion process; bulk resources
- **Michigan Medical Equipment:** Conversion process; identifiable resources, labor
- **Nantasket Basket, Gasket, and Casket:** Conversion cycle, material acquisition; multiple employees
- **Vivian’s Fashion Factory II:** Acquisition, conversion, and revenue

We took the twelve models listed above and remodeled them one by one with our REA-DSL tool. For each of the models, we started off by creating the agent view and resource view. In a second step, we analyzed the events and commitments in the original model and defined the according value activities in the value chain view. Last, depending on the original model structure, we defined operational views and/or planning views in the REA-DSL tool. During this process we came across some concepts our REA-DSL did not support at this time. We also discussed the results with Bill McCarthy and incorporated all the requests and missing concepts to our REA-DSL. Consequently, in a last run we were able to successfully remodel all twelve REA-DSL models without the loss of any business semantics. Notice, we did not consider some specific policy relationships (e.g., policies directly between agents and resources) and business locations. This requires to add a separate *Policy View* to the REA-DSL, which is part of future work. Furthermore, we did not consider some concepts in the models, which are not part of REA.

While remodeling the original REA class-like example with our REA-DSL tool, we already notice some benefits of our approach. First of all, the different views allow a quick overview of the different REA concepts at a glance instead of searching them in the class diagram. Second, the value chain gives the modeler a quick overview of the value activities and a good starting point for modeling the exchanges and conversions. Third, in most cases, by automatically generating the planning/operational stubs from the value activity, we received a model that was close to the finished one. In most cases, we only had to rename agents and add some additional agents to conform to the REA class example. Forth, if the operational and planning view are conceptually congruent, we do not need to model the operational view separately, since the semantic for the operational view is already implicitly given by the planning view. Consequently, the REA class-like representation results for this exchange/conversion in twice as many elements.

Last, we bring one example with conceptually congruent REA layers and compare the complexity between the class-like representation and the REA-DSL. In Figure 8.6 you see the decrement sale event type and commitment on the planning layer of the REA-DSL. In Figure 8.7 you see the corresponding example in the class-like REA representation.

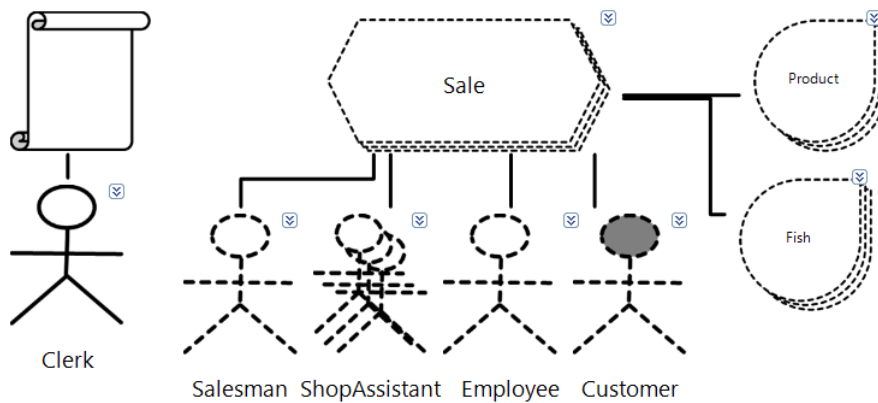


Figure 8.6: Complexity REA-DSL Sale

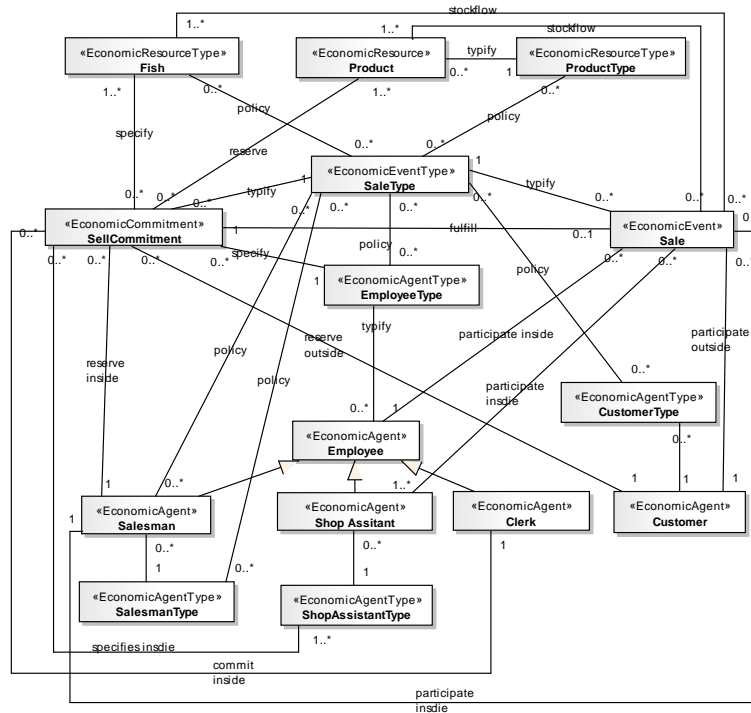


Figure 8.7: Complexity REA Class-Like Sale

In the REA-DSL we were able to keep the same semantics with 9 elements instead of 15 elements in the class-like representation. This is a reduction by 40 percent without any semantic loss. Regarding the relationships between the elements we were able to reduce the 27 relationships of the class-like representation to only 7 relationships in the REA-DSL model. This is an astonishing reduction by 74 percent.

8.3 Expert Interviews

According to Bogner [Bog05], there are three different kinds of expert interviews: explorative, systematic, and theory-generating expert interviews. In our evaluation, we make use of a mixture of explorative and systematic expert interviews. Whereas explorative expert interviews help to gain basic knowledge in a new or diffuse domain, the systematic expert interview tries to answer specific questions about the domain at hand. The expert can be seen as an advisor to provide valuable knowledge about this domain. In our case this domain is the REA ontology and we conducted many rounds of interviews with different experts. In the early stages of this thesis, we gained an overview of the basic REA ontology. In latter stages, we had more specific structured questions to sharpen the knowledge about the REA ontology, especially on the concepts of the other two layers (i.e., planning layer and policy layer). In the following, we present the different experts we worked with as well as the various conferences and talks we participated in.

In the course of this thesis we scheduled weekly Skype calls with the founder of REA, Bill McCarthy. We discussed progress of our REA-DSL and incorporated the valuable feedback into the REA-DSL. Additionally, Bill McCarthy also provided many additional ideas we added to the tool. Most important, we were able to clarify the meaning of the sometimes vague described concepts of the REA ontology. Furthermore, we met twice for in person meetings – one time at the TU Vienna in Austria and one time for a week at the Michigan State University (MSU). These meetings were especially important, since we were able to dwell in detail on the twelve REA class diagram examples introduced in the previous section. This gave us a lot of insight into the REA ontology.

Within the TU Vienna we have five faculty members familiar with the REA ontology. Interim results were shown to these people and ideas for improvement were discussed.

Besides presenting our REA-DSL at international conferences (CAiSE 2011, ICEBE 2011), we also participated in the *5th and 6th International Workshop on Value Modeling and Business Ontology* (VMBO 2011, VMBO 2012). The two latter workshops were the perfect community for discussing our results, since more than half of the participants are experts in the REA ontology. Furthermore, we were invited to give a presentation of the REA ontology in the USA at the *Semantic Modeling of Accounting Phenomena* (SMAP 2011) workshop to present the results to the accounting community. Last, we also held a presentation at a *Accounting Information Systems* class at MSU.

We gained a lot of valuable feedback from the community by the actions taken above which we used to improve the REA-DSL. In general, we are confident to say that the REA experts appreciated our approach.

8.4 Usability Study

The final evaluation we performed on our REA-DSL was a usability study, where 16 probands which are either familiar with the REA ontology or with conceptual models participated. We evaluated human factors criteria [How95] such as the user-friendliness and the comprehensiveness [GAM05] of the two different REA representations – class-like REA models and REA-DSL models, respectively. Therefore we also followed ideas from evaluating domain-specific mod-

eling solutions by Mohagheghi et al. [MH10]. They define different stakeholders of domain-specific modeling solutions, from which we consider *domain experts* (knowledge about REA) and *software engineers* (knowledge about conceptual modeling) as relevant for our approach.

Procedure. In the beginning, we explained the probands the basic procedure of the usability study and provided them a document (cf. Appendix A). First, we taught them the basic REA concepts and how they are represented by the class diagrams as well as by the REA-DSL showing them a small example of *Sy's Fish* (cf. Appendix A.1). For the actual evaluation we prepared one example REA model in the class like representation (a pizza selling company) and one example REA model in our REA-DSL notation (a computer selling company). Both examples have a similar structure considering buying a resource, assembling a final product, and selling a final product (cf. Appendix A.2). The probands were asked to look at the class-like REA model example and answer various questions in the given order on a questionnaire (cf. Appendix A.3). The first part of the questionnaire contains questions testing if and how fast the proband understood business relevant parts of the REA class-like model. Therefore, the time needed by the probands for each question was recorded. Second, the probands were asked to look at the REA-DSL model and answer the same questions in the given order for the REA-DSL model as for the class-like REA model. Accordingly, the time the probands needed for each question was recorded again. We had about half of the probands (7) look at the class-like example first and about half of the probands (9) look at the REA-DSL example first. The last part of the questionnaire contains qualitative questions about the general comprehensiveness of the REA-DSL and of the class-like REA model. These questions ask which features are better in the REA-DSL representation or in the class-like representation and additionally, for the perceived ease of use (EOU) as described by the technology acceptance model (TAM) by Davis [Dav89].

Interpretation of Results. Figure 8.8 shows the average time in seconds users needed to answer each of the ten questions regarding the REA business model. The blue bars show the time users needed to answer the questions for the class-like REA representation, whereas the red bars show the time users needed to answer the questions for the REA-DSL representation. These times also includes the time the probands needed to write down the answers. The time to actually write each answer for the REA-DSL and for the class-like representation is about the same since business case of the examples are of the same complexity. In average, all the ten questions were faster answered when looking at the REA-DSL representation than looking at the class-like representation. Questions 1, 2, 3, 4, 7, and 9 were answered at least twice as fast. We should notice here, that the order of the questions might influence the time needed to answer, because after each question the proband gained knowledge about the complete example.

When looking at the cumulated time of the users needed to answer the ten questions (cf. box plot in Figure 8.9) we can see that the median time answering the ten questions for the REA class-like representation was 619 seconds (10 minutes and 19 seconds), whereas answering the same questions for the REA-DSL representation was 238 seconds (3 minutes and 58 seconds). This is a time reduction by about 2.6 times. The maximum time for answering the questions for the REA-DSL (407 seconds) was still lower than the minimum time answering the questions for the class-like representation (425 seconds). For the REA-DSL, the first quartile is at 202

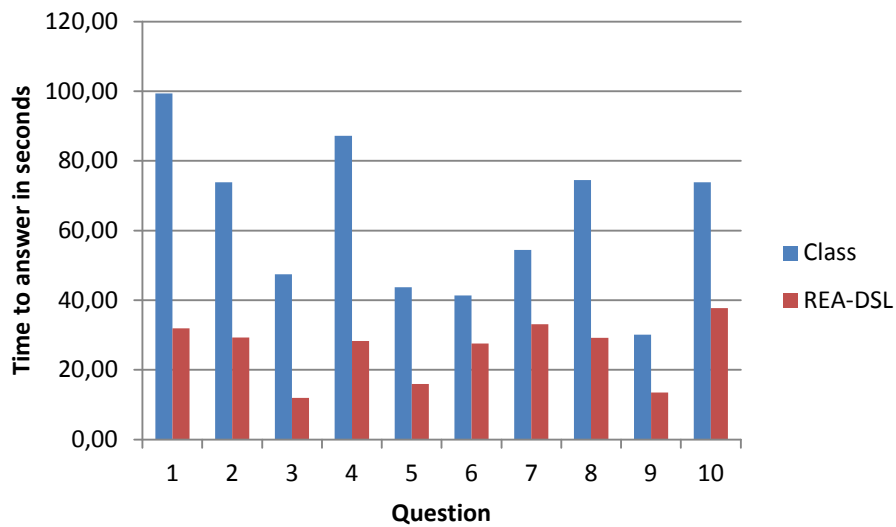


Figure 8.8: Time Used to Answer REA Model Questions

seconds and the third quartile is at 318 seconds, whereas for the class-like representation the first quartile is at 497 seconds and the third quartile at 743 seconds. Even though this is a preliminary usability evaluation of our REA-DSL, the results are promising and underpin our Hypothesis 1, that a domain-specific language for REA is easier to understand.

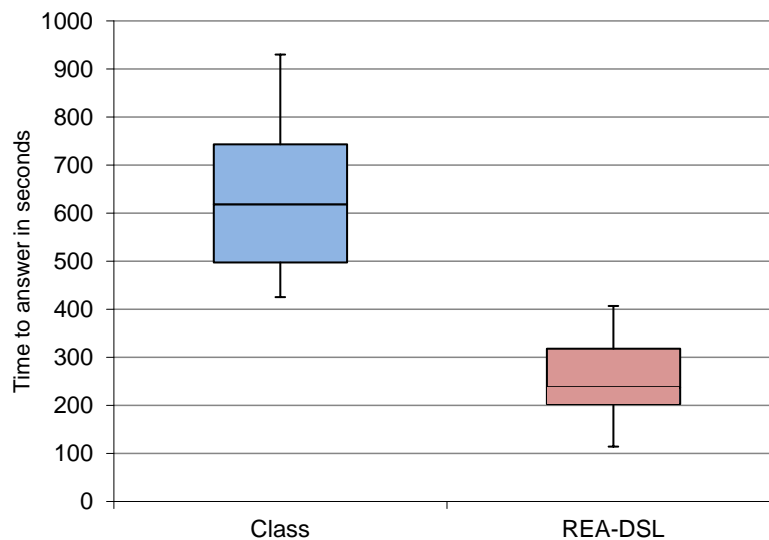


Figure 8.9: Cumulated Time Used to Answer REA Model Questions

In Figure 8.10 the percentage of wrong answers for each question for either the REA-DSL or class-like representation is given. We can see, that in the class-like representation 50 % of

the probands did not answer question 2 (“What are the resources of the company”) completely right. Mostly, they forgot at least one resource. We argue, that it is harder to find all resources which are depicted in the same way as other REA concepts in the class-like representation, than to find them in the REA-DSL, where there is one specific notation for a resource. Question 3 (“Which resource is not uniquely identifiable”) was answered right for the REA-DSL by all probands, but was answered wrong by 30 % for the class-like representation. In the class-like representation, one has to figure out if there exists a resource additionally to the resource type or not. If not, it is not uniquely identifiable. This requires a lot of scanning of the class-diagram and fosters overseeing certain classes. In the REA-DSL, not uniquely identifiable resources can be easily seen by looking at the dedicated representation of a dashed drop. Furthermore, concerning question 7 (“Who commits to the commitment fulfilled by a future pizza/computer sale”), more than 40 % of the class-like representation answers and more than 30 % of the REA-DSL answers were wrong. In those cases, the probands also mentioned the agent of the reciprocal commitment additionally to the actual committing agent. For all the other questions not specifically mentioned here, the error rate was below 30 %. Overall, in average 21 % of the questions for the class-like representation were answered wrong and in average about 8 % of the REA-DSL questions were answered wrong.

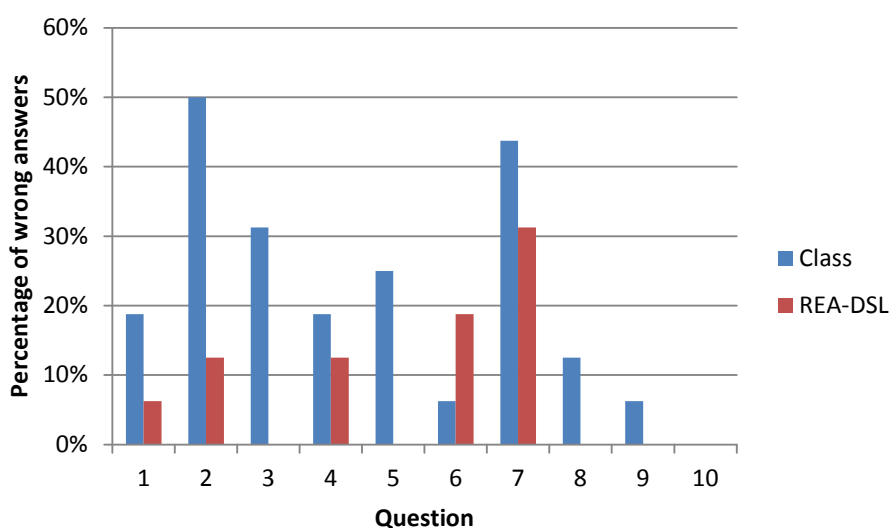


Figure 8.10: Wrong Answers Given

Last, Figure 8.11 shows the average answers given to the user preference questions (cf. general questions in Appendix A.3). Each question could be answered with either 5 - strongly agree, 4 - agree, 3 - undecided, 2 - disagree, and 1 - strongly disagree. In the following, we reflect on each question/statement which are additionally marked by their number in brackets.

The probands strongly agreed, that (1) the different notation elements for the different REA concepts of the REA-DSL are easier to understand than one common class representation. This is a major benefit we were able to address with domain-specific languages. In general, the participants strongly agree, that (2) the differentiation of the inside and outside agent is easier to

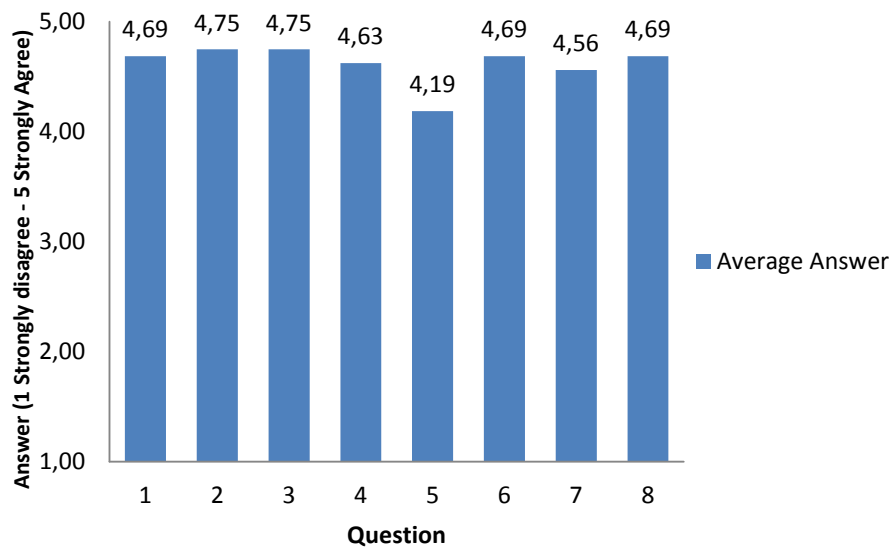


Figure 8.11: Preferences of Users

understand by black and white stick figures than by participate-inside and participate-outside relationships of the class-like representation. We believe, that this stick figures can be easier detected than the relationships and their annotation. Next, the users strongly agreed that (3) the main business processes can be detected easier with the REA-DSL. We argue, that this is due to the dedicated representation of the business processes in a separate view. In a class-like representation, the participants had to look for the duality relationship in the diagram to find the business processes which takes more time as shown by our timed questions. Most participants stated, that (4) the different views of the REA-DSL make REA models more clear than one common class diagram. We believe, that the different views lessen the overall complexity and split the business model into smaller parts. The statement (5) that using staples of agents/resources for multiple agents/resources than annotating relationships with 1..* is more intuitive provided us with various feedback. In average, the probands agree, however it shows that this features is subject to further discussion. Additionally, at workshops we received feedback, where REA experts told us that they like the 1..* representation better. However, we believe that for pure business people the stacks are more intuitive. All participants strongly agreed, that (6) the REA-DSL seems to be less complex than the class representation. We think, that this is due to the different facts of having separate views and dedicated notation elements. In general, the probands strongly agreed that (7) the REA-DSL is easier to use and understand which summarizes the answers given before. Last, the participants strongly agreed, that (8) they would use the REA-DSL instead of the class-like representation for modeling REA.

8.5 Hypotheses Verification

In this chapter, we presented a first evaluation of our REA-DSL which supports the hypotheses declared in the beginning of this thesis:

1. **Hypothesis 1.** *A domain-specific language of the REA ontology is easier to understand than the class-like representation.*

This hypothesis is verified by the usability study and expert interviews we conducted. First of all, the usability study has shown, that for answering similar business questions the average time is less than half when using the REA-DSL compared to using the REA class-like representation. Furthermore, fewer wrong answers were given using the REA-DSL. The positive feedback from the expert interviews and the business modeling community on comprehensiveness of our REA-DSL underpin these results.

2. **Hypothesis 2.** *A domain-specific language of the REA ontology allows an unambiguous creation of REA ontology business models.*

The verification of this hypothesis is provided by a functional test as well as by the expert interviews. For the functional test, we had twelve structurally different REA class-like models, which cover the majority of REA concepts, at hand. We modeled each single one of them with the REA-DSL. Afterwards, we proved that we were able to keep all semantics in the REA-DSL models. This result is also underpinned by the expert interviews with Bill McCarthy, where in several video conference calls we refined the REA-DSL to integrate all important REA concepts.

3. **Hypothesis 3.** *A domain-specific language of the REA ontology can be mapped to a database scheme for an accounting information system.*

This hypothesis is verified by the technical feasibility of our REA-DSL designer tool. After creating the REA-DSL models, the tool automatically generated the corresponding relational database structures by means of SQL statements. We loaded the relational database structure into a relational data modeling tool and verified that all concepts were correctly mapped to the relational model in compliance with the REA ontology.

Conclusion and Future Work

The REA ontology is a powerful accounting framework and might be able to replace the more than 500 year old tradition of double-entry bookkeeping in the accounting discipline. REA provides a business modeling ontology for describing the economic phenomena of an enterprise inside as well as outside of its borders. Its strong basis on literature in economic theory and its ability to define not only events, resources, and agents but also commitments and policies sounds promising for being used in the design phases of an accounting information system. However, REA has not received wide adoption so far, which we argued may be due to vague formalization, missing graphical representations, and missing automation for creating REA models. IT professionals designing the AIS system as well as business people having the knowledge about the company's activities could benefit from overcoming these limitations.

Thus, in this thesis, we presented a *business model driven data engineering* approach based on a newly created domain-specific language called REA-DSL for designing an accounting information systems (AIS). Instead of dealing with cumbersome and vague class represented REA models, the REA-DSL provides an unambiguous and dedicated graphical notation. Furthermore, it is built upon a meta-model, which provides a proper formalization of the REA-DSL. Furthermore, a serialization format is provided enabling model-exchange. A mapping between the REA-DSL and a relational database schema enables the use of the conceptual REA model for the underlying database of an AIS.

The missing dedicated graphical representation and the vague formalization has also been criticized by Gailly and Poels who have proposed a new conceptual representation of REA guided by proven ontology engineering principles [GP07a]. They define a presentation based on UML class diagrams as depicted in Figure 9.1. It shows a commitment for selling fish and products in return for cash. Furthermore, all the involved agents are shown. The example of the REA class diagram is semantically equivalent to our REA-DSL shown in Figure 9.2. We feel, that the class-like representation results in a complex visual representation that is difficult to understand by domain experts. Thus, it is hard for business experts to define a conceptual model for an AIS with the class-like REA representation.

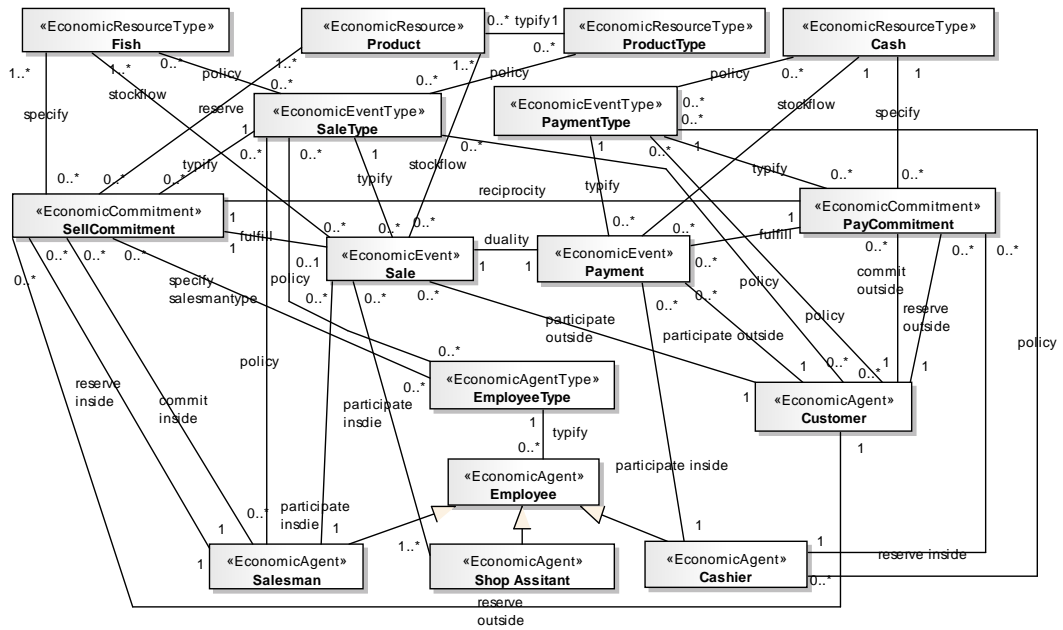


Figure 9.1: REA Class Example

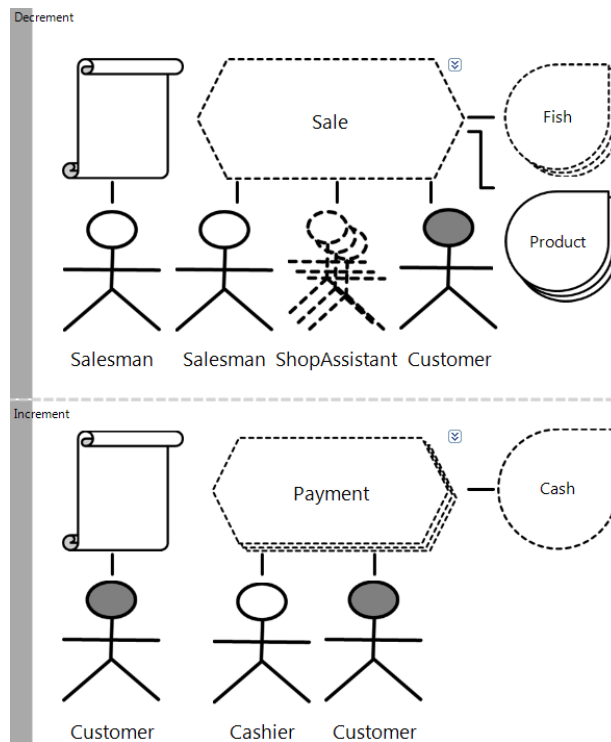


Figure 9.2: REA-DSL Example

We argue that our REA-DSL can overcome these limitations. Consequently, the main research question for the *business model driven data engineering* approach of this thesis is:

”How can we provide a precise and easy to understand modeling language for the REA business modeling ontology which may be used for the semi-automatic development of accounting information systems?”

This question is tackled by various contributions, which we reflect on in the following.

9.1 Contributions

Our proposed REA-DSL is a domain-specific modeling language supporting the conceptual modeling of economic events. Such conceptual models aim at facilitating the requirements elicitation process during the design of accounting and enterprise information systems. The REA-DSL builds upon the three different layers of the REA ontology: the operational layer, the planning layer, and the policy layer. The operational layers allows capturing current and past events together with their exchanged resources and their participating agents. On the planning layer, commitments and agreements can be defined for these events. The policy layer is used to define policies and standards for events, agents, and resources.

REA-DSL formalization. Our first contribution is the specification of the formalization format for the REA-DSL that inhibits divergent interpretations. In particular, the relationships between the concepts of the three REA layers are precisely defined by using the *Object Management Group’s* (OMG) *Meta Object Facility* (MOF) leading to a dedicated REA domain-specific modeling language. The REA-DSL is formalized by an unambiguous meta-model for the REA ontology. The meta-model defines, how the REA concepts relate to each other. For example, it defines that resources are connected to events through stockflow relationships and that agents are connected to events by participation relationships. Additionally, constraints applied on this meta-model assure the correct usage of the REA ontology. Furthermore, rules defined on the meta-model put additional constraints according to the REA ontology on the meta-model.

REA-DSL representation. The next contribution is a dedicated graphical syntax for our REA-DSL covering a set of stencils that facilitate the understanding of the domain expert. At the same time, the REA-DSL remains powerful enough to express all concepts of the REA ontology. Consequently, we replaced the class representation of the general purpose language UML formerly used for the REA ontology by a domain-specific language. Instead of using the same rectangle class shape for every REA concept, we defined different shapes for different concepts. Resources are depicted by the shape of a drop, events are depicted by the shape of a hexagon, and agents are depicted by stick figures. These concepts can be connected to each other and form a valid REA ontology model. Another important concept we introduced in the REA-DSL is the bulk resource. A bulk resource is a resource, which cannot be individually identified (e.g., water, oil) or one where it is not economically feasible to track individually (e.g., nails, chewing gums).

To lessen the complexity of the REA models, we introduced five interlinked REA-DSL views. In the agent view all the agents and their generalization hierarchies are defined. Similar,

in the resource view all the resources and their generalization hierarchies are defined. The value chain view provides an overview of all the transfers and transformations of resources of a company. This provides a good starting point for defining the actual transfers and transformations, which are further defined in the operational view by events.

REA-DSL commitments and policies. Commitments on these transfers/transformations can additionally be defined in separate planning view models. The planning view allows describing these commitments and contracts for future events. They are depicted by scrolls and are connected to the committing agent. When the actual event happens, these commitments get fulfilled. The planning view makes use of the three additional artifacts agent type, resource type, and event type. The importance of these type artifacts are demonstrated by the following example: at the time of the commitment, the information of the identifiable agent/resource might not be available. Thus, a commitment might only specify a certain type of agent or resource. This specified agent type or resource type becomes an identifiable agent or identifiable resource at the time of the fulfilling event.

The planning view also allows deriving basic policy structures between events, agents, and resources. Policies allow describing certain standards or restrictions for the events, resources, and agents. For example, policies can define which agent is allowed to participate in which events, or it can define the standard price for a resource.

REA-DB extensions and relational mapping. The problem that REA-DSL models could not be processed and read by IT systems in order to automatically create database structures, is tackled by our contribution of the REA-DB extensions for the REA-DSL. Without this extension, the IT professional has to perform the error prone and time intensive task of remodeling the REA-DSL to a relational model, which can be understood by database systems. Consequently, we presented the extension of the REA-DSL by the database concepts attributes and primary keys, and provide mapping rules to a relational model.

Therefore, in a first step, we had to extend the various REA concepts by properties. For example, properties for a resource might be its color or weight. Furthermore, commitment properties allow capturing specifics such as the committed price of a resource in a sale. Especially the policy properties deliver a concept to define policies, standards, and constraints between other REA concepts. If the property is the uniquely identifying property of the element, it can be marked as the primary key. The properties are represented in the graphical notation by compartments containing the properties. There are three different kind of compartments: a green compartment holding all the object properties, a yellow compartment holding all the type and policy properties, and a purple compartment holding all the commitment properties. Primary keys are annotated by a small key symbol.

In a second step, we contribute a mapping between the extended REA-DSL and a relational database schema, which can be used by the IT expert as the basis for an AIS. The mapping consists of four sub-mappings: a resource view to relational mapping, an agent view to relational mapping, an operational view to relational mapping, and a planning view to relational mapping. All these mappings are defined by pseudo codes. Object hierarchies in the REA models are mapped to suitable representations in the relational model. We used SQL as the data definition language to describe the target relational models since it is widely supported.

REA-XML serialization. As another contribution, we presented the REA-XML, an unambiguous and tool-independent language for REA models. Given its name REA-XML is based on XML and, thus, corresponds to an XML Schema as a textual, non-graphical notation for REA models. Using XML as a serialization syntax for REA models offers the potential (i) to use any XML editor for specifying REA models, (ii) store the models in a registry for business models, and (iii) to use it as an exchange language between different REA tools. Its human-readable XML format would even allow viewing the REA-DSL models in a text editor (obviously, without the comfort of the graphical representation). Consequently, we defined a transformation between the REA-DSL format and the REA-XML language. According to the REA-DSL, the created REA-XML schema consists of five different parts: the agent view XSD, the resource view XSD, the operational view XSD, the planning view XSD, and the value chain view XSD. All these five views are combined by the REA XSD. We especially see the opportunity that with the help of the serialization format models are more likely to be stored in a registry and can be reused by other business modelers with similar business cases.

9.2 Evaluation Summary

The evaluation builds upon four main pillars: (i) the technical feasibility of the REA-DSL as well as of the mapping to the relational model was demonstrated by a tool implementation, (ii) a functional test based on existing REA models was performed, (iii) expert interviews were taken, most notable with the founder of REA, Bill McCarthy, and (iv) a usability study with IT and REA professionals was conducted. All four evaluation parts delivered promising results and verified the hypotheses stated in the introduction. The tool implementation allows modeling a complete REA-DSL model with assisting wizards and generated stubs. After the REA-DSL model is designed, it can be automatically transformed by code generation templates to a relational database schema based on SQL. A relational data modeling tool can import this schema or a relational database system of an AIS can use this schema to create its database structure. After applying the complete mapping on the accompanying example Sy's Fish with its seven value activities, 103 tables conforming to the REA ontology were generated. With the finalized REA-DSL tool we aimed at making the AIS development process faster and more streamlined. We were even asked to provide a copy of our tool in order to be used for educational purposes. However, the tool implementation is still on a prototypical level and some more work needs to be done to get it ready for industrial and extended educational use. Additionally, the experience and positive as well as critical feedback gained from the expert interviews can be used for further extensions and improvements of the REA-DSL to reach our vision of a model-driven AIS development platform.

9.3 Limitations and Future Work

Even though the REA-DSL presented in this thesis is a powerful graphical modeling language for the REA ontology, capturing many economic phenomena and enabling derivation of data structures for AIS data bases, it is unquestionable that there are still many challenges ahead for the REA community. First of all, there is no doubt, that the provided tool still resembles a pro-

totypical implementation merely for evaluation purposes. But we intend to enhance the tool in order to make it available not only to academics but also to industry. We see a high potential for additional extensions of the existing REA-DSL as well as the integration of the REA-DSL with other modeling languages and methodologies. Concluding this thesis, we provide an outlook for future work the REA-DSL may benefit from.

REA further extensions. The REA-DSL introduced in this thesis spans over the three REA ontology layers: the operational layer, the planning layer, and the policy layer. Concerning the policy layer, we incorporated the typification specifications and basic policy structures of this layer. The typifications allow specifying yet-not-known agents or resources in a commitment. Policy relationships may be defined between event types and resources (types), or between events and agents (types). These policy relationships allow defining standards, restrictions, and blue prints for events and commitments. However, we did not include the policy relationships between resources and agents or allow modeling more complex policy structures [GM06]. For example, such advanced policies may define what pilot is allowed to fly which type of an airplane or what vendor can provide which resource. Thus, these concepts need to be included into an extended REA-DSL. We imagine, that this can be done by an additional REA-DSL view called the policy view, allowing the definition of additional relations between the REA concepts. In addition, various authors have suggested minor extensions to REA to deal with context specific details (e.g., [Hru06, HHKV09, SA11]). Additionally, business locations might be added to the REA-DSL defining at what location events are happening or at what locations resources are located. These proposals are candidates to be included as extension to the REA-DSL.

A neutral REA perspective. In the area of Web Services and business processes distinction is often made between (i) orchestration representing a process from the perspective of a participant who is the central controller and (ii) choreography representing a peer-to-peer process without central controller from the neutral perspective of an observer (cf. [Pel03]). A similar distinction seems to be useful for REA as well. All the basic REA papers, such as [McC82, GM97, GM06], are based on the local view of a participant. Likewise, our REA-DSL is as well based on the participant's view. An observer's perspective on the interactions between different parties – as outlined in [ISO07] and [Lau10] - is not supported. This perspective would allow a neutral view on the partners' network using the REA ontology showing commitments between partners and unveiling resource flows between them.

Mapping to other data models. The mapping provided in this thesis is between the REA-DSL and a relational logical model. This is particular useful, since it keeps the control of the created relational tables in our hands. In future work these mapping could be extended in order to incorporate different generalization mapping strategies which can be chosen before the actual mapping is executed. However, we also feel the need for creating mappings between our REA-DSL and conceptual data models such as Entity-Relationship diagrams or class diagrams. This would allow modelers to modify the data model on a higher conceptual level. According to the preferences of modelers, they could then chose to either generate conceptual models or logical relational data models from the REA-DSL.

Terms and exceptions. REA exception-handling has rarely been studied in the literature so far. REA papers usually describe “happy path” scenarios where all parties fulfill their commitments. In order to reflect business reality, it is necessary to address the exceptional “penalty

paths” when some parties do not fulfill their commitments. Imagine ordering a fish and it is not delivered at the right time specified in the commitment. In this case terms apply to certain commitments and specify additional events such as penalty payments or additional provided discounts. Consequently, terms need to be integrated into the REA-DSL adding legal aspects to the REA-DSL.

Reference models and patterns. REA actually exists of many recurring patterns. In order to speed up the development process, REA business patterns [Hru06] such as the account pattern, the due date pattern, the notification pattern, etc. might be considered for the REA-DSL. To integrate the patterns into the REA-DSL, techniques from the area of reference modeling, such as configuration, instantiation, specification, aggregation, and analogy, might be used [vB07, Tho05,FL07]. We discuss the idea of possibly integrating REA among other modeling languages with reference modeling in [HHK⁺12, May10].

Workflows, phases, and states. The current REA-DSL lacks a concept to define the required sequences for commitments and events. Workflows added to the REA-DSL may specify this order. They might be based on the five different Open-edi business transaction phases specified in the ISO/IEC 15944-1 standard [ISO11b]: planning, identification, negotiation, actualization, and post-actualization. In addition, ideas of Horiuchi and McCarthy [HM11] deriving the essential business objects and their state changes caused by the REA operations might be used.

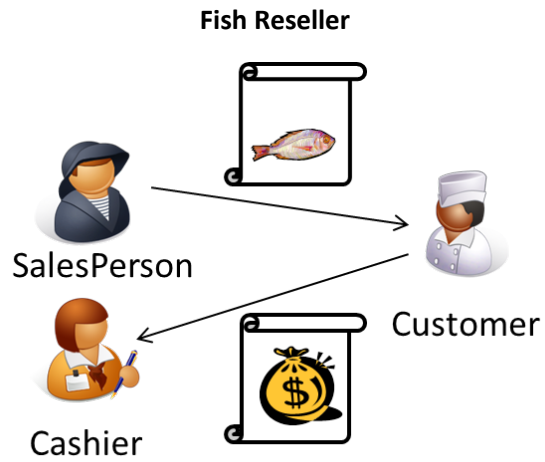
Integrating REA with e³value and BPMN in a model-driven enterprise information system approach. To the best of our understanding, REA should not only be considered as an isolated approach towards accounting information systems. We feel that REA may reach its full potential when becoming part of a suite of approaches for enterprise information systems. When eliciting the requirements for an enterprise information system, one has to address three layers: the first layer addresses the business model or the value perspective describing by which means a company makes profit. A well-recognized candidate on this level is e³value [GA01]. The second layer addresses contracts, agreements, and commitments as well as internal schedules that are required to realize the business model. REA perfectly fits the needs of this layer. The third level addresses the business processes describing the steps/activities to execute the contracts, agreements, and commitments as well as the internal schedules. There exists a plethora of business process modeling notations, among which the BPMN has recently gained a lot of attraction. BPMN would be a good replacement for all the system flowcharting conventions now commonly used in accounting. Accordingly, it would be helpful integrating REA with e³value and BPMN within a model driven approach towards the requirements of enterprise information systems. Having considered all these relationships with other methods, it would be possible to describe a REA based development process for the requirements elicitation of enterprise information systems.

Usability Study Documents

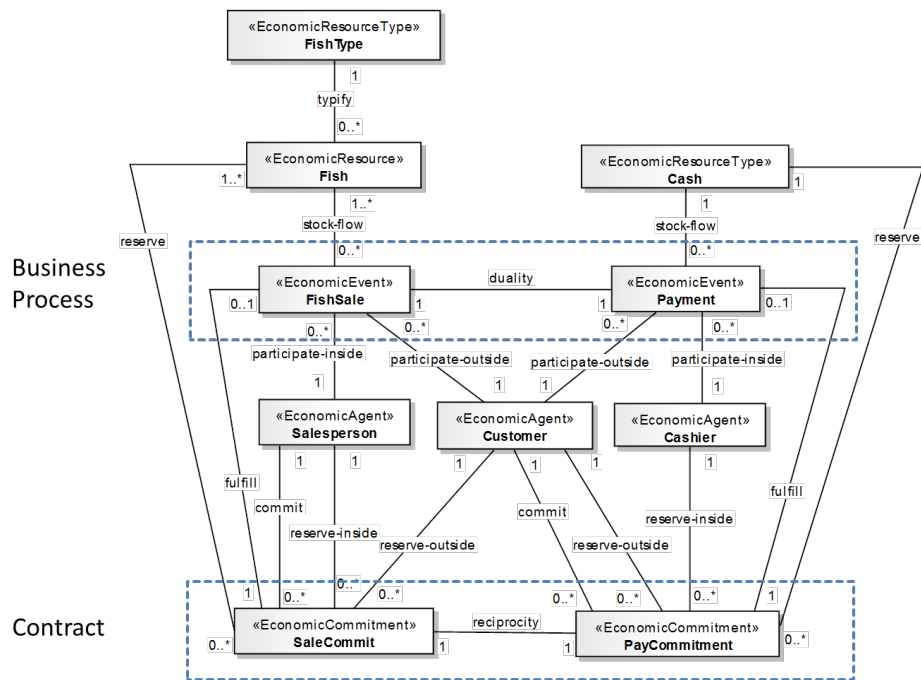
On the following six pages the documents provided to the probands for the usability study are shown. The first two pages (cf. Appendix A.1) contain the fish case example which was used to get the probands familiar with REA and the two different notations - the REA class-like representation and the REA-DSL representation. The proceeding two pages (cf. Appendix A.2) show the class-like and REA-DSL example the probands had to answer questions for. The last two pages (cf. Appendix A.3) show the questions asked after the probands had to look at the REA class-like and REA-DSL evaluation examples.

A.1 Teaching Examples

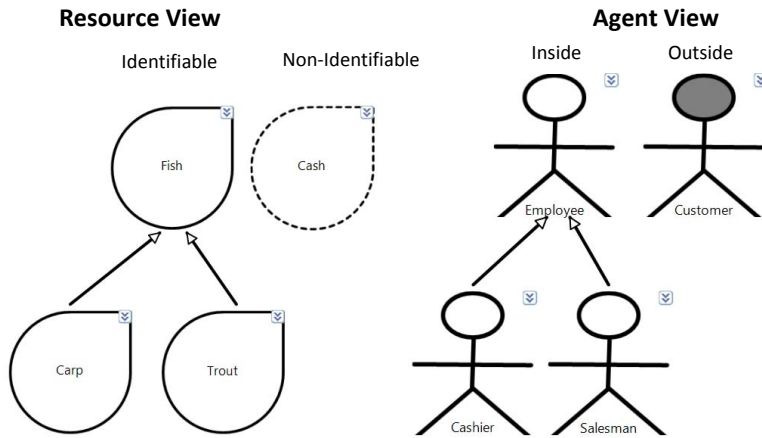
Use case example



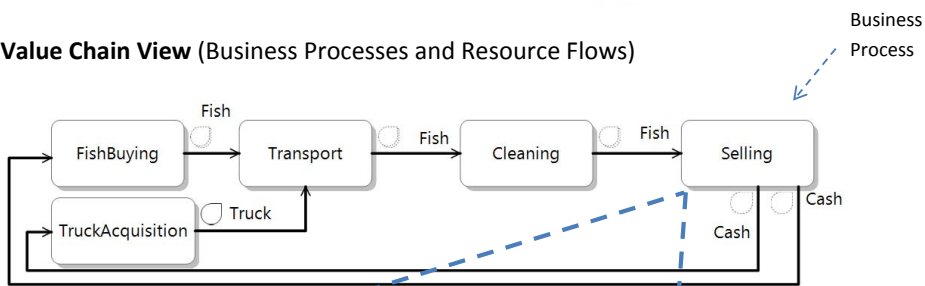
REA Class Representation



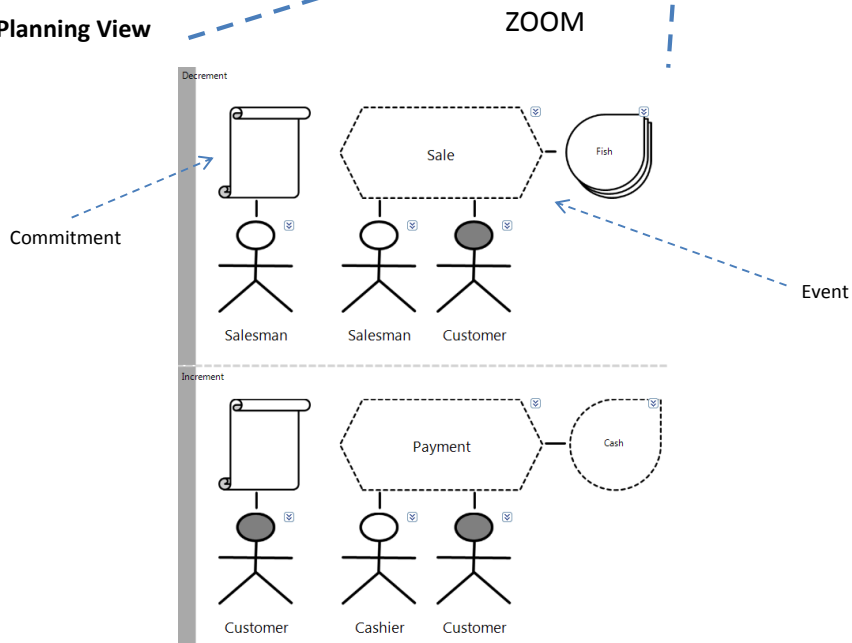
REA-DSL Representation



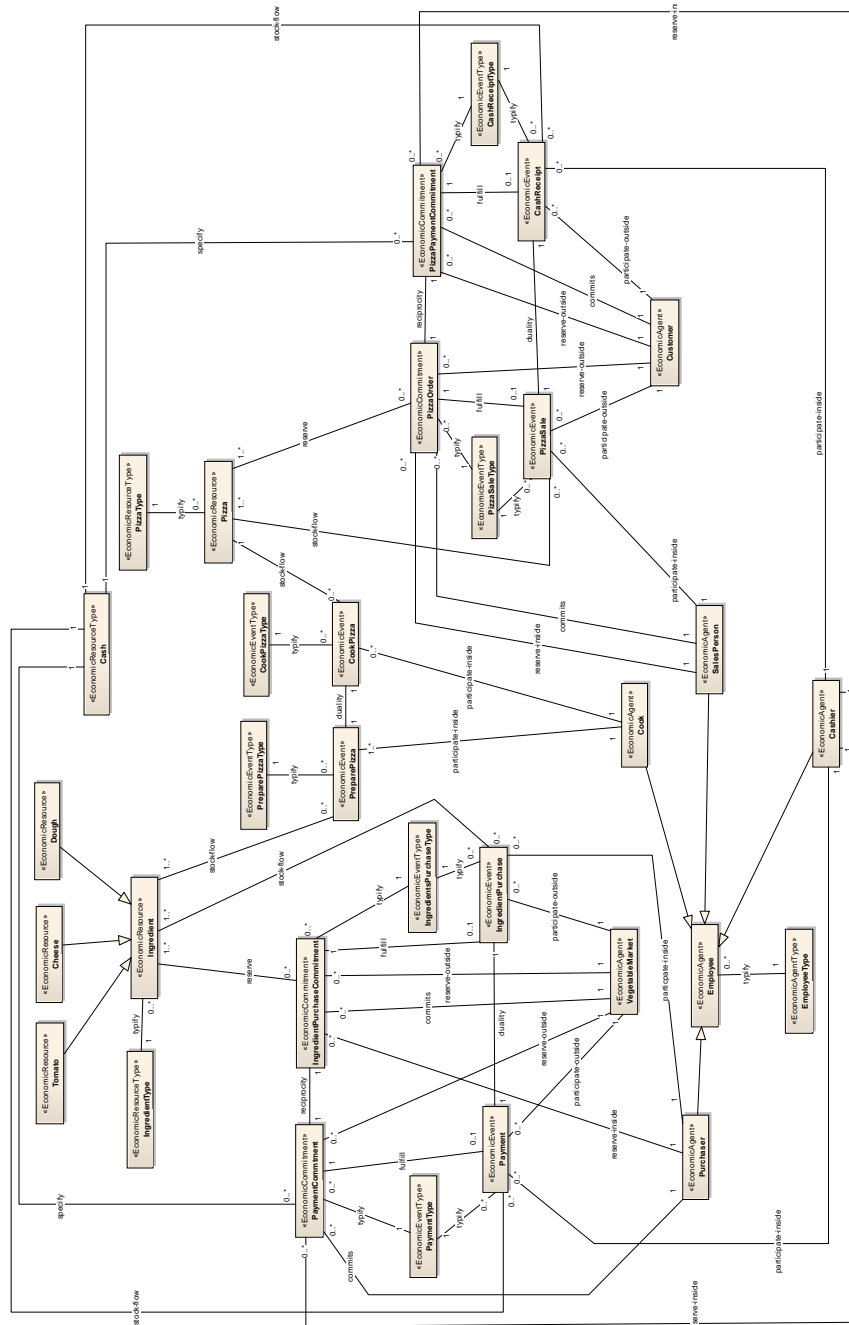
Value Chain View (Business Processes and Resource Flows)



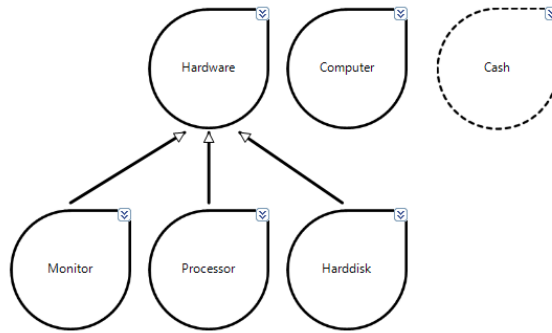
Planning View



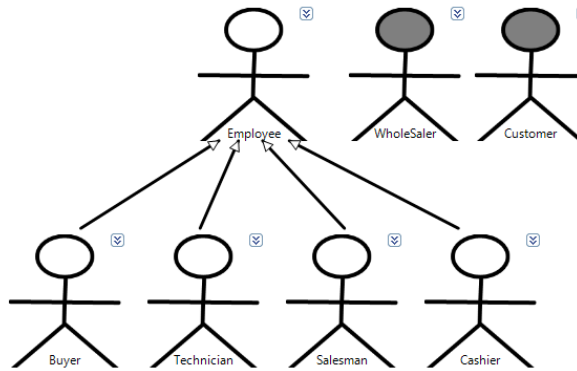
A.2 Evaluation Examples



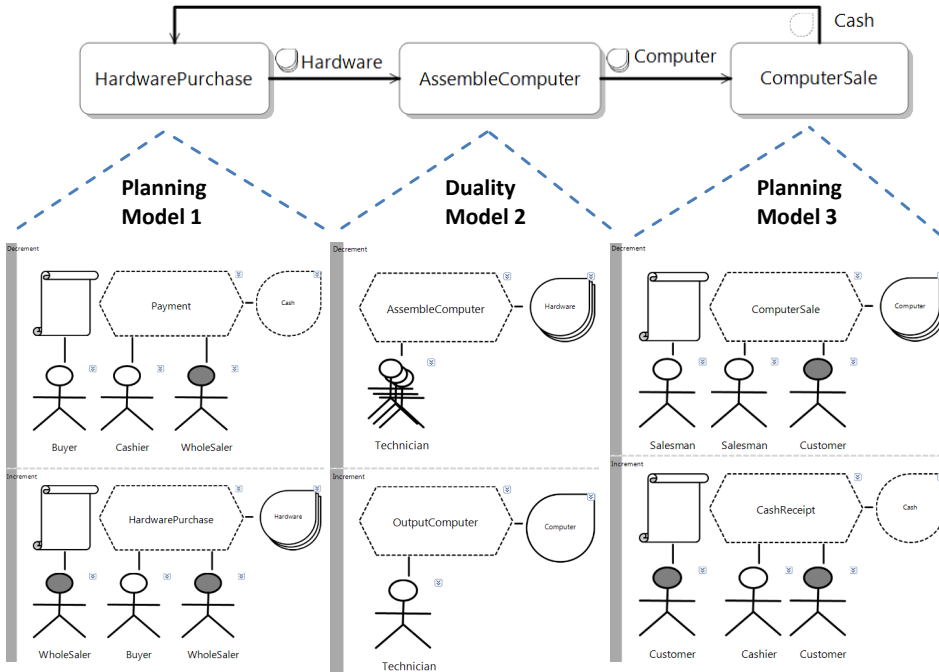
Resource Model



Agent Model



ValueChain



A.3 Questionnaire

Questionnaire REA class-like versus REA-DSL

Name: DSL first: REA/CM knowledge:

REA class-like timed questions (Time in MM:SS)

1. What are the business processes of the company? (__ : __)
2. What are the resources of the company? (__ : __)
3. Which resource is not uniquely identifiable? (__ : __)
4. Who are the agents **inside** the company? (__ : __)
5. Who are the agents **outside** of the company? (__ : __)
6. Who participates in the pizza sale event? (__ : __)
7. Who commits to the commitment fulfilled by a future pizza sale? (__ : __)
8. In what events is cash involved? (__ : __)
9. Can multiple pizzas be sold in a pizza sale event? (__ : __)
10. In the pizza preparing/cooking business process, what resources are consumed and what resources are created? (__ : __)

REA-DSL timed questions (Time in MM:SS)

1. What are the business processes of the company? (__ : __)
2. What are the resources of the company? (__ : __)
3. Which resource is not uniquely identifiable? (__ : __)
4. Who are the agents **inside** the company? (__ : __)
5. Who are the agents **outside** of the company? (__ : __)
6. Who participates in the computer sale event? (__ : __)
7. Who commits to the commitment fulfilled by a future computer sale? (__ : __)
8. In what events is cash involved? (__ : __)
9. Can multiple computers be sold in a computer sale event? (__ : __)
10. In the assembling computer business process, what resources are consumed and what resources are created? (__ : __)

General questions

1. The different notations (e.g., stick figures, drops) for the different REA concepts are easier to understand than one common class representation
 Strongly agree Agree Undecided Disagree Strongly Disagree
2. Differentiation of the inside and outside agents by black and white stick figures of the REA-DSL is easier to understand than by the participate-inside and participate-outside relationships of the class-like representation
 Strongly agree Agree Undecided Disagree Strongly Disagree
3. The main business processes can be detected easier with the REA-DSL
 Strongly agree Agree Undecided Disagree Strongly Disagree
4. The different views of the REA-DSL make REA models more clear than one common class-diagram
 Strongly agree Agree Undecided Disagree Strongly Disagree
5. Using staples of agents/resources for multiple agents/resources in the REA-DSL is more intuitive than annotating the relationship with 1..* in the class diagram
 Strongly agree Agree Undecided Disagree Strongly Disagree
6. The REA-DSL seems to be less complex than the class representation
 Strongly agree Agree Undecided Disagree Strongly Disagree
7. In general, the REA-DSL is easier to use and understand
 Strongly agree Agree Undecided Disagree Strongly Disagree
8. For modeling REA, I would use the REA-DSL instead of the class-like representation
 Strongly agree Agree Undecided Disagree Strongly Disagree

OCCL Constraints

B.1 Operational View OCL Constraints

In a transfer, at least one inside and exactly one outside agent participates in each event.

context Transfer

inv: self.incrementset.union(self.decrementset).events.
forAll(e | e.participations->exists(p | p.agent.isKindOf(InsideAgent)))

context Transfer

inv: self.incrementset.union(self.decrementset).events.
forAll(e | e.participations->one(p | p.agent.isKindOf(OutsideAgent)))

In a transformation, at least one inside agent participates in each event and no outside agents participates in an event.

context Transformation

inv: self.incrementset.union(self.decrementset).events.
forAll(e | e.participations->exists(p | p.agent.isKindOf(InsideAgent)))

context Transformation

inv: self.incrementset.union(self.decrementset).events.
forAll(e | not e.participations->exists(p | p.agent.isKindOf(OutsideAgent)))

B.2 Planning View OCL Constraints

In a contract, at least one inside agent (type) and exactly one outside agent (type) participates in each event.

context Contract

inv: self.incrementplan.union(self.decrementplan).eventtypes.
forAll(e | e.reserveparticipations->exists(p | p.agent.isKindOf(InsideAgent)
or p.agenttype.isKindOf(InsideAgentType)))

context Contract

inv: self.incrementplan.union(self.decrementplan).eventtypes.
forAll(e | e.reserveparticipations->one(p | p.agent.isKindOf(OutsideAgent)
or p.agenttype.isKindOf(OutsideAgentType)))

In a schedule, at least one inside agent (type) participates in each event and no outside agent (type) participates in an event.

context Schedule

inv: self.incrementplan.union(self.decrementplan).eventtypes.
forAll(e | e.reserveparticipations->exists(p | p.agent.isKindOf(InsideAgent)
or p.agenttype.isKindOf(InsideAgentType)))

context Schedule

inv: self.incrementplan.union(self.decrementplan).eventtypes.
forAll(e | not e.reserveparticipations->exists(p | p.agent.isKindOf(OutsideAgent)
or p.agenttype.isKindOf(OutsideAgentType)))

No agent series is allowed to commit.

context Commitment

inv: not self.commits.agent.ocllsTypeOf(InsideAgentSeries)

context Commitment

inv: not self.commits.agent.ocllsTypeOf(OutsideAgentSeries)

In a contract only inside agents can commit on decrement commitments and outside agents on increment commitment.

context Contract

inv: self.decrementplan.commitlayer.commitment.commits.agent.
ocllsTypeOf(InsideAgent)

context Contract

inv: self.incrementplan.commitlayer.commitment.commits.agent.ocIsTypeOf(OutsideAgent)

In a schedule only inside agents can commit on commitments.

context Schedule

inv: self.decrementplan.commitlayer.commitment.commits.agent.ocIsTypeOf(InsideAgent)

context Schedule

inv: self.incrementplan.commitlayer.commitment.commits.agent.ocIsTypeOf(InsideAgent)

List of Figures

1.1	Accounting Information System (AIS) Design	3
1.2	REA-DB: Contributions of this Thesis	6
1.3	Knowledge Contribution Framework [Hev09]	13
2.1	State of the Art - Overview	15
2.2	DSL Approach	18
2.3	e ³ value Model	20
2.4	Value Chain Notations	22
2.5	DSL Approach	26
2.6	Object-Relational Mapping Strategies	29
3.1	REA Core Concepts	32
3.2	Vertical and Horizontal REA Layers [GM02] (extended)	34
3.3	Type Model Layers	37
3.4	REA UML Profile [UN/08,Sch10]	39
4.1	Sy's Fish Overview	46
4.2	REA Meta-Model	47
4.3	Sy's Fish Object Constellation for Restaurant Fish Sale	48
4.4	Sy's Fish Object Constellation for Restaurant Fish Order	49
4.5	REA Sy's Fish Example	50
5.1	MOF Levels Applied on the REA-DSL	53
5.2	REA Views	54
5.3	Resources Notation Elements	55
5.4	Agents Notation Elements	56
5.5	Event/Commitment/Value Activity Notation Elements	57
5.6	Association Notation Elements	58
5.7	Agent Meta-Model	59
5.8	Agent Abstract Model	59
5.9	Resource Meta-Model	61
5.10	Resource Abstract Model	61
5.11	Operational View Meta-Model	64
5.12	Operational View Abstract Example	65

5.13	Labor Acquisition with Labor Resource	66
5.14	Value Chain View Meta-Model	68
5.15	Value Chain View Abstract Example	69
5.16	Value Chain Multiple Resource to Event Single Resource	71
5.17	Planning View Meta-Model	73
5.18	Planning View Abstract Example	74
5.19	Labor Acquisition Event Type with Labor Resource	76
5.20	Policy Example	78
5.21	Policy Meta-Model	78
5.22	Sy's Fish Agent View	80
5.23	Sy's Fish Resource View	80
5.24	Sy's Fish Value Chain View	81
5.25	Sy's Fish Operational View	82
5.26	REA all Operational Views	83
5.27	Sy's Fish Planning View	84
5.28	Sy's Fish Planning View Product Alternative	85
5.29	REA Example all Planning Views	86
6.1	Agent Properties Meta-Model	88
6.2	Resource/Bulk Resource Properties Meta-Model	89
6.3	Event/Event Type Properties Meta-Model	90
6.4	Stockflow and Participate Properties Meta-Model	91
6.5	Agent Mapping	94
6.6	Resource Mapping	96
6.7	Bulk Resource Mapping	96
6.8	Labor Resource Mapping	98
6.9	Operational Mapping	99
6.10	Operational Stockflow and Participate Mapping	101
6.11	Planning Mapping	102
6.12	Planning Stockflow and Participate Mapping	104
7.1	REA-DSL to XML Mapping	108
7.2	REA XSD Parts	108
7.3	Agent Meta-Model	109
7.4	Agent XSD	110
7.5	Sy's Fish Agent Model	111
7.6	Resource Meta-Model	112
7.7	Resource XSD	113
7.8	Sy's Fish Resource Model	114
7.9	REA Duality Meta-Model	115
7.10	Duality XSD	116
7.11	Sy's Fish Duality Model	119
7.12	Value Chain Meta-Model	120
7.13	Value Chain XSD	121
		171

7.14	Sy's Fish Value Chain Model	122
7.15	REA Planning Meta-Model	124
7.16	Planning XSD	125
7.17	Sy's Fish Selling Planning Model	127
8.1	DSL Approach	130
8.2	Excerpt of the Agent Domain Model	131
8.3	T4 Template to SQL Statements to Relational Model	133
8.4	REA-DSL Tool	134
8.5	REA-DSL Modeling Steps	136
8.6	Complexity REA-DSL Sale	143
8.7	Complexity REA Class-Like Sale	143
8.8	Time Used to Answer REA Model Questions	146
8.9	Cumulated Time Used to Answer REA Model Questions	146
8.10	Wrong Answers Given	147
8.11	Preferences of Users	148
9.1	REA Class Example	152
9.2	REA-DSL Example	152

Nomenclature

AIS Accounting Information System
BMO Business Model Ontology
DSL Domain-Specific Language
EMF Eclipse Modeling Framework
ERD Entity-Relationship Diagram
ERP Electronic Resource Planning
IS Information System
MOF Meta-Object Facility
REA Resource-Event-Agent Ontology
SDK Software Development Kit
TAM Technology Acceptance Model
T4 Text Template Transformation Toolkit
UML Unified Modeling Language
UMM UN/CEFACT's Modeling Methodology
XMI XML Metadata Interchange
XML Extensible Markup Language

Bibliography

- [ABE⁺06] Birger Andersson, Maria Bergholtz, Ananda Edirisuriya, Tharaka Ilayperuma, Paul Johannesson, Jaap Gordijn, Bertrand Grégoire, Michael Schmitt, Eric Dubois, Sven Abels, Axel Hahn, Benkt Wangler, and Hans Weigand. Towards a Reference Ontology for Business Models. In *Proceedings of the 25th International Conference on Conceptual Modeling (ER 2006)*, Tucson, Arizona, USA, pages 482–496. Springer, 2006.
- [Ale02] John Alexander. History of Accounting. Association of Chartered Accountants in the United States <http://documents.clubexpress.com/documents.ashx?key=7ZPfhrqSH4ej5qOo06gTZ1j%2FWfzYw%2BhpXBNOQ%2BbRiWgYV1UQpbPezRxbi%2FPDVo7X>, 2002.
- [AT01] Allan Afuah and Christopher L. Tucci. *Internet Business Models and Strategies: Text and Cases*. McGraw-Hill Higher Education. Irwin/McGraw-Hill, 2001.
- [AZ01] Raphael Amit and Christoph Zott. Value Creation in E-business. *Strategic Management Journal*, 22(6-7):493–520, 2001.
- [Ben86] Jon Bentley. Programming Pearls: Little Languages. *Communications of the ACM*, 29(8):711–721, 1986.
- [Bod83] George H. Bodnar. *Accounting Information Systems*. Allyn & Bacon, Inc., 1983.
- [Bog05] Alexander Bogner. *Das Experteninterview: Theorie, Methode, Anwendung (in German)*. VS Verlag für Sozialwissenschaften, 2005.
- [BPS94] Michael Blaha, William Premerlani, and Hwa Shen. Converting OO Models into RDBMS Schema. *Software, IEEE*, 11(3):28–39, 1994.
- [BSN09] Nancy A. Bagranoff, Mark G. Simkin, and Carolyn Strand Norman. *Core Concepts of Accounting Information Systems*. John Wiley and Sons, 11th edition, 2009.
- [Bus11] Business Process Technology Research Group, Hasso-Plattner-Institute. *Oryx*, <http://bpt.hpi.uni-potsdam.de/Oryx>, 2011.
- [CB74] Donald D. Chamberlin and Raymond F. Boyce. SEQUEL: A Structured English Query Language. In *Proceedings of 1974 ACM-SIGMOD Workshop on Data Description, Access and Control*, pages 249–264, 1974.

- [CC05] Luca Cabibbo and Antonio Carosi. Managing Inheritance Hierarchies in Object/Relational Mapping Tools. In *Proceedings of the 17th International Conference on Advanced Information Systems Engineering (CAiSE 2005), Porto, Portugal*, pages 135–150. Springer, 2005.
- [Che76] Peter Pin-Shan Chen. The Entity-Relationship Model - Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1:9–36, 1976.
- [CI06] Janie Chang and Laura Ingraham. *Modeling and Designing Accounting Systems: Using Access to Build a Database*. John Wiley and Sons, New York, NY, USA, 2006.
- [CJKW07] Steve Cook, Gareth Jones, Stuart Kent, and Alan Cameron Wills. *Domain-Specific Development with Visual Studio DSL Tools*. Addison-Wesley, 2007.
- [CMO94] Jia-Lin Chen, Dennis McLeod, and Daniel O’Leary. Schema Evolution for Object-based Accounting Database Systems. In *Proceedings of the International Symposium on Object-Oriented Methodologies and Systems (ISOOMS 1994), Palermo, Italy*, pages 40–52. Springer, 1994.
- [CNYM00] Lawrence Chung, Brian A. Nixon, Eric Yu, and John Mylopoulos. *Non-Functional Requirements in Software Engineering*. Kluwer Academic Press, Inc., 2000.
- [Cod70] Edgar Codd. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13:377–387, 1970.
- [CPKK06] Krzysztof Czarnecki, Chang Hwan Peter Kim, and Karl Trygve Kalleberg. Feature Models are Views on Ontologies. In *Proceedings of the 10th International Conference on Software Product Lines (SPLC 2006), Baltimore, Maryland, USA*, pages 41–51. IEEE, 2006.
- [Dat03] Chris J. Date. *An Introduction to Database Systems*. Addison-Wesley-Longman, 8th edition, 2003.
- [Dav89] Fred D. Davis. Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. *MIS Quarterly*, 13(3):319–340, 1989.
- [DB07] Rob Davis and Eric Brabänder. *ARIS Design Platform: Getting Started with BPM*. Springer, 2007.
- [DCH05] Cheryl L. Dunn, J. Owen Cherrington, and Anita S. Hollander. *Enterprise Information Systems: A Pattern-Based Approach*. McGraw-Hill Irwin, 2005.
- [DD97] Chris Date and Hugh Darwen. A Guide to SQL Standard, 4th edition. 1997.
- [DM97] Cheryl L. Dunn and William E. McCarthy. The REA Accounting Model: Intellectual Heritage and Prospects for Progress. *Journal of Information Systems*, 11(1):31–51, 1997.

- [Dom12] Domain-Specific Modeling Forum. DSM Tools. <http://www.dsmforum.org/tools.html/>, 2012.
- [EN10] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems, 6th edition*. Addison-Wesley-Longman, 2010.
- [FL07] Peter Fettke and Peter Loos. Perspectives on Reference Modeling. In Peter Fettke and Peter Loos, editors, *Reference Modeling for Business Systems Analysis*, pages 1–20. Idea, 2007.
- [FP96] George T. Friedlob and Franklin J. Plewa. *Understanding Balance Sheets*. John Wiley and Sons, 1996.
- [FV95] Christian Fahrner and Gottfried Vossen. A Survey of Database Design Transformations Based on the Entity-Relationship Model. *Data & Knowledge Engineering*, 15(3):213–250, 1995.
- [GA01] Jaap Gordijn and Hans Akkermans. E3-value: Designing and Evaluating E-Business Models. *IEEE Intelligent Systems*, 16(4):11–17, 2001.
- [GA03] Jaap Gordijn and Hans Akkermans. Value Based Requirements Engineering: Exploring Innovative E-Commerce Idea. *Requirements Engineering Journal*, 8(2):114–134, 2003.
- [Gai10] Frederik Gailly. Conceptual Modeling Using Domain Ontologies: Improving the Domain-Specific Quality of Conceptual Schemas. In *Proceedings of the 10th Domain-Specific Modelling Workshop (DSM 2010), in conjunction with the 25th Conference on Object-Oriented Programming, Systems, Languages, and Applications (SPLASH/OOPSLA 2010)*, number 2009/573. University Ghent, 2010.
- [GAM05] Martin Grossman, Jay E. Aronson, and Richard V. McCarthy. Does UML Make the Grade? Insights from the Software Development Community. *Information and Software Technology*, 47(6):383–397, 2005.
- [GEPP08] Frederik Gailly, Sergio Espana, Geert Poels, and Oscar Pastor. Integrating Business Domain Ontologies with Early Requirements Modelling. In *Proceedings of the Workshop on Advances in Conceptual Modeling: Challenges and Opportunities, in conjunction with the 27th International Conference on Conceptual Modeling (ER2008)*, pages 282–291, 2008.
- [GHL⁺11] Christoph Grün, Christian Huemer, Philipp Liegl, Dieter Mayrhofer, Thomas Motal, Rainer Schuster, Hannes Werthner, and Marco Zapletal. eBusiness. In *Handbook of Semantic Web Technologies*, pages 787–848. Springer, Berlin, Heidelberg, 2011.
- [GLP07] Frederik Gailly, Wim Laurier, and Geert Poels. Positioning REA as a Business Domain Ontology. In *Proceedings of the REA-25 Conference, Newark, Delaware, USA*, 2007.

- [GLP08] Frederik Gailly, Wim Laurier, and Geert Poels. Positioning and Formalizing the REA Enterprise Ontology. *Journal of Information Systems*, 22(2):219–248, 2008.
- [GM86] Graham Gal and William E. McCarthy. Operation of a Relational Accounting System. *Advances in Accounting*, 3:83–112, 1986.
- [GM97] Guido L. Geerts and William 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, 1997.
- [GM99] Guido L. Geerts and William E. McCarthy. An Accounting Object Infrastructure for Knowledge-Based Enterprise Models. *IEEE Intelligent Systems*, 14(4):89–94, 1999.
- [GM00a] Guido L. Geerts and William E. McCarthy. Augmented Intensional Reasoning in Knowledge-Based Accounting Systems. *Journal of Information Systems*, 14(1):127–150, 2000.
- [GM00b] Guido L. Geerts and William E. McCarthy. The Ontological Foundations of REA Enterprise Systems, Alabama. <http://www.msu.edu/user/mccarth4/Alabama.doc>, 2000.
- [GM01] Guido L. Geerts and William 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.
- [GM02] Guido L. Geerts and William E. McCarthy. An Ontological Analysis of the Economic Primitives of the Extended-REA Enterprise Information Architecture. *International Journal of Accounting Information Systems*, 3(1):1–16, 2002.
- [GM04] Guido L. Geerts and William E. McCarthy. Type-Level Specification in REA Enterprise Systems (Working Paper). <https://www.msu.edu/user/mccarth4/UTS-seminar/Type%20paper%20final%20submission.doc>, 2004.
- [GM05] Guido L. Geerts and William E. McCarthy. The Ontological Foundations of REA Enterprise Systems, Tulane. <http://www.msu.edu/user/mccarth4/tulane.doc>, 2005.
- [GM06] Guido L. Geerts and William E. McCarthy. Policy-Level Specification in REA Enterprise Information Systems. *Journal of Information Systems*, 20(2):37–63, 2006.

- [GMR96] Guido L. Geerts, William E. McCarthy, and Stephen R. Rockwell. Automated Integration of Enterprise Accounting Models Throughout the Systems Development Life Cycle. *Intelligent Systems in Accounting, Finance and Management*, 5(3):113–128, 1996.
- [GOP05] Jaap Gordijn, Alexander Osterwalder, and Yves Pigneur. Comparing Two Business Model Ontologies for Designing E-Business Models and Value Constellations. In *Proceedings of the 18th BLED eConference (e-Integration in Action)*, Bled, Slovenia. University of Maribor, 2005.
- [Gor11] Jaap Gordijn. The E3-value Toolset. <http://www.e3value.com>, 2011.
- [GP07a] Frederik Gailly and Geert Poels. Ontology-Driven Business Modelling: Improving the Conceptual Representation of the REA Ontology. In *Proceedings of the 26th International Conference on Conceptual Modeling (ER 2007)*, Auckland, New Zealand, pages 407–422. Springer, 2007.
- [GP07b] Frederik Gailly and Geert Poels. Towards Ontology-Driven Information Systems: Redesign and Formalization of the REA Ontology. In *Proceedings of the 10th International Conference on Business Information Systems (BIS 2007)*, pages 245–259, 2007.
- [GP09] Frederik Gailly and Geert Poels. Using the REA Ontology to Create Interoperability between E-Collaboration Modeling Standards. In *Proceedings of the 21st International Conference on Advanced Information Systems Engineering (CAiSE 2009)*, 2009.
- [GR99] Asunción Gómez-Pérez and Dolores Rojas. Ontological Reengineering and Reuse. In *Proceedings of the 11th European Knowledge Acquisition Workshop (EKAW 1999)*, Dagstuhl Castle, Germany. Springer, 1999.
- [Gru93] Thomas 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.
- [GSCK08] Jack Greenfield, Keith Short, Steve Cook, and Stuart Kent. *Software Factories*. John Wiley and Sons, 2008.
- [GY08] Shouping Gui and Zhishen Ye. The Research of Value Chain Operation Model in E-Commerce. In *Proceedings of the International Seminar on Business and Information Management (ISBIM 2008)*, Wuhan, Hubei, China, pages 238–241. IEEE, 2008.
- [Har07] Paul Harmon. *Business Process Change: A Guide for Business Managers and BPM and Six Sigma Professionals*. The MK/OMG Press. Elsevier/Morgan Kaufmann Publishers, 2007.

- [HC93] Michael Hammer and James Champy. Reengineering the Corporation: A Manifesto for Business Revolution. *Business Horizons*, 36(5):90–91, 1993.
- [HDC99] Anita Hollander, Eric Denna, and J. Owen Cherrington. *Accounting, Information Technology, and Business Solutions*. McGraw-Hill Higher Education, 2nd edition, 1999.
- [Hev09] Alan R. Hevner. Design Science Research in Information Systems, talk at Vienna University of Technology, Trends in E-Commerce Lecture. http://www.ec.tuwien.ac.at/files/u326/Vienna_2011_DSR_Presentation_hevner.pdf, 2009.
- [HHK⁺12] Birgit Hofreiter, Christian Huemer, Gerti Kappel, Dieter Mayrhofer, and Jan vom Brocke. Inter-organizational Reference Models - May Inter-organizational Systems Profit from Reference Modeling? In *both, Business System Management and Engineering in Springer LNCS volume 7350 and Proceedings of the International Workshop on Business System Management and Engineering (BSME 2010), in conjunction with the 48th International Conference on Objects, Models, Components, Patterns (TOOLS 2010), Malaga, Spain*, pages 1–16. Springer, 2012.
- [HHKV09] Frantisek Hunka, Miroslav Hucka, Josef Kasik, and Dominik Vymetal. Enterprise Planning Model Using REA Ontology. In *Proceedings of the 3rd International Workshop on Value Modeling and Business Ontologies (VBMO 2009), Stockholm, Sweden*, 2009.
- [HHKV10] Frantisek Hunka, Miroslav Hucka, Josef Kasik, and Dominik Vymetal. REA Value Chain Applied on Production Planning Model. In *Proceedings of the 2nd International Conference on Logistics and Transport, Christchurch, New Zealand*, 2010.
- [HM11] Satoshi Horiuchi and William E. McCarthy. An Ontology-Based State Machine for Catalog Orders. In *Proceedings of the 5th International Workshop on Value Modeling and Business Ontologies (VMBO 2011), Ghent, Belgium*, 2011.
- [HMPR04] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design Science in Information Systems Research. *MIS Quarterly*, 28(1):75–105, 2004.
- [How95] James Howatt. A Project-Based Approach to Programming Language Evaluation. *ACM SIGPLAN Notices*, 30:37–40, 1995.
- [Hru05] Pavel Hruby. Ontology-Based Domain-Driven Design. In *Proceedings of the Workshop on Best Practices for Model-Driven Software Development in conjunction with the 20th Annual Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2005), San Diego, CA, USA*. ACM, 2005.
- [Hru06] Pavel Hruby. *Model-Driven Design Using Business Patterns*. Springer, 1st edition, 2006.

- [Iji75] Yuji Ijiri. *Theory of Accounting Measurement*. American Accounting Association, Sarasota, Fla, 1975.
- [Ila07] Tharaka Ilayperuma. *Reference Ontology for Business Models*. PhD thesis, Stockholm University, Sweden, 2007.
- [ISO07] ISO. *Information Technology - Business Operational View - Part 4: Business Transaction Scenarios: Accounting and Economic Ontology*, ISO/IEC 2007, ISO 15944-4, http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=40348, 2007.
- [ISO11a] ISO. *Information technology – Database languages – SQL – Part 1: Framework (SQL/Framework)*, ISO/IEC 9075-1:2008, http://www.iso.org/iso/catalogue_detail.htm?csnumber=45498, 2011.
- [ISO11b] ISO. *Information technology - Business Operational View - Part 1: Operational aspects of Open-edi for implementation*, ISO/IEC 2002, ISO 15944-1, http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=55289, 2011.
- [Kap00] Raphael Kaplinsky. *A Handbook for Value Chain Research*. University of Sussex Institute of Development Studies, 2000.
- [KC09] Vahid R. Karimi and Donald D. Cowan. Verification of Access Control Policies for REA Business Processes. In *Proceedings of the 33rd Annual International Computer Software and Applications Conference (COMPSAC 2009), Seattle, Washington, USA*, volume 2, pages 422–427. IEEE, 2009.
- [KN05] Robert S. Kaplan and David R. Norton. The Balanced Scorecard: Measures That Drive Performance. *Harvard Business Review*, 83(7/8):172–180, 2005.
- [Kro05] David Kroenke. *Database Processing: Fundamentals, Design, and Implementation*. Prentice Hall India, 8th edition, 2005.
- [KT08] Steven Kelly and Juha-Pekka Tolvanen. *Domain-Specific Modeling*. Wiley-IEEE Computer Society Press, 2008.
- [Lau10] Wim Laurier. *The Resource-Event-Agent Ontology as a Foundation for Business Modeling*. PhD thesis, Ghent University, Belgium, 2010.
- [LC00] Jane Linder and Susan Cantrell. Changing Business Models: Surveying the Landscape. *Accenture - Institute for Strategic Change*, <http://course.shufe.edu.cn/jpkc/zhanlue/upfiles/edit/201002/20100224120954.pdf>, 2000.
- [Lea10] Neal Leavitt. Will NoSQL Databases Live Up to Their Promise? *Computer*, 43(2):12–14, 2010.

- [Lee99] Allen S. Lee. Remarks from MIS Quarterly Editor - Inaugural Editor's Comments. *MIS Quarterly*, 23(1), 1999.
- [LG07] Fakhar Lodhi and Muhammad Ahmad Ghazali. Design of a Simple and Effective Object-to-Relational Mapping Technique. In *Proceedings of the 2007 Symposium on Applied Computing (SAC2007), New York, NY, USA*, pages 1445–1449. ACM, 2007.
- [LM09] Philipp Liegl and Dieter Mayrhofer. A Domain Specific Language for UN/CE-FACT's Core Components. In *Proceedings of the International Workshop on Service Computing for B2B (SC4B2B 2009) in conjunction with the 5th World Congress on Services (SCC 2009), Bangalore, India*, pages 123–131. IEEE, 2009.
- [LP07] Wim Laurier and Geert Poels. Three Fundamental REA Business Patterns. In *Proceedings of the REA-25 Conference - Special Track on REA Business Patterns, Newark, Delaware, USA*, page 24, 2007.
- [LW94] Barbara Liskov and Jeannette M. Wing. A Behavioral Notion of Subtyping. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(6):1811–1841, 1994.
- [Man11] Management Informatics, Ghent University. *Resource Event Agent Enterprise Ontology Version 2*, http://www.managementinformatics.ugent.be/cgi-bin/php-cgiwrap/a64526/REAv2-WIKI/index.php?title=Main_Page, 2011.
- [May10] Dieter Mayrhofer. Reference Modeling for Inter-organizational Systems. In *Proceedings of the MODELS Doctoral Symposium, in conjunction with the 13th International Conference on Model Driven Engineering Languages and Systems (MODELS 2010), Oslo, Norway*, pages 37–42. ACM/IEEE, 2010.
- [McC79] William E. McCarthy. An Entity-Relationship View of Accounting Models. *The Accounting Review*, pages 667–686, 1979.
- [McC80] William E. McCarthy. Construction and Use of Integrated Accounting Systems with Entity-Relationship Modelling. In *Proceedings of the 1st International Conference on the Entity-Relationship Approach to Systems Analysis and Design*, pages 625–637. ACM, 1980.
- [McC82] William E. McCarthy. The REA Accounting Model: A Generalized Framework for Accounting Systems in a Shared Data Environment. *The Accounting Review*, 57(3):554–578, 1982.
- [McC03] William E. McCarthy. The REA Modelling Approach to Teaching Accounting Information Systems. *Issues in Accounting Education*, 18(4):427–441, 2003.

- [MH10] Parastoo Mohagheghi and Øystein Haugen. Evaluating Domain-Specific Modelling Solutions. In Juan Trujillo, Gillian Dobbie, Hannu Kangassalo, Sven Hartmann, Markus Kirchberg, Matti Rossi, Iris Reinhartz-Berger, Esteban Zimányi, and Flavius Frasincar, editors, *Advances in Conceptual Modeling - Applications and Challenges*, volume 6413 of *Lecture Notes in Computer Science*, pages 212–221. Springer, 2010.
- [MH12a] Dieter Mayrhofer and Christian 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.
- [MH12b] Dieter Mayrhofer and Christian 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.
- [MHHS11] Dieter Mayrhofer, Christian Huemer, Birgit Hofreiter, and Christian Sonnenberg. REA-XML: An Unambiguous Language for REA Business Models. In *Proceedings of the 8th IEEE International Conference on e-Business Engineering (ICEBE 2011)*, Beijing, China, pages 1–8. IEEE, 2011.
- [MHS05] Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and How to Develop Domain-Specific Languages. *ACM Computing Surveys*, 37(4):316–344, 2005.
- [Mic11a] Microsoft. *T4 Text Template Transformation Toolkit*, <http://msdn.microsoft.com/en-us/library/bb126445.aspx>, 2011.
- [Mic11b] Microsoft. *Visual Studio 2010 Visualization and Modeling SDK (VMSDK)*, <http://archive.msdn.microsoft.com/vsvmsdk>, 2011.
- [MM02] Ibrahim Mohamed and Salwa Mansor. Semantic Modelling For School Information System (SIS). In *Proceedings of the 10th International Workshop on Software Technology and Engineering Practice (STEP 2002)*, Washington, DC, USA, pages 114–117. IEEE, 2002.
- [MRG07] Maddeh Mohamed, Mohamed Romdhani, and Khaled Ghedira. MOF-EMF Alignment. In *Proceedings of the 3rd International Conference on Autonomic and Autonomous Systems (ICAS 2007)*, Washington, DC, USA, pages 1–6. IEEE, 2007.
- [MS09] Thomas Motal and Rainer Schuster. From E3-Value to REA: Modeling Multi-Party eBusiness Collaborations. In *Proceedings of the 11th Conference on Commerce and Enterprise Computing (CEC 2009)*, Vienna, Austria, pages 202–208. IEEE, 2009.

- [MSHH11] Dieter Mayrhofer, Christian Sonnenberg, Birgit Hofreiter, and Christian Huemer. A Domain Specific Modeling Language for REA. In *Proceedings of the 5th International Workshop on Value Modeling and Business Ontology (VMBO 2011)*, Ghent, Belgium, 2011.
- [Ö07] Turhan Özgür. Comparison of Microsoft DSL Tools and Eclipse Modeling Frameworks for Domain-Specific Modeling In the context of the Model-Driven Development. Master’s thesis, Blekinge Institute of Technology, Sweden, 2007.
- [Obj10] Object Management Group. *XML Metadata Interchange (XMI), Version 2.4*. OMG, <http://www.omg.org/spec/XMI/2.4/Beta2/PDF>, 2010.
- [Obj11a] Object Management Group. Meta Object Facility (MOF) Core Specification, Version 2.4. <http://www.omg.org/spec/MOF/2.4/Beta2>, 2011.
- [Obj11b] Object Management Group. *Unified Modeling Language (UML): Infrastructure 2.4*, <http://www.omg.org/spec/UML/2.4/Infrastructure/Beta2/PDF>, 2011.
- [Obj11c] Object Management Group. *Unified Modeling Language (UML): Superstructure 2.4*, <http://www.omg.org/spec/UML/2.4/Superstructure/Beta2/PDF>, 2011.
- [Obj11d] Object Management Group. *Value Delivery Modeling Language (VDML)*, <http://neffics.eu/wp-content/uploads/2011/06/11-05-11-VDM.pdf>, 2011.
- [Obj12] Object Management Group. *Object Constraint Language (OCL), Version 2.3.1*. OMG, <http://www.omg.org/spec/OCL/2.3.1/PDF>, 2012.
- [O’L04] Daniel E. O’Leary. On the Relationship Between REA and SAP. *International Journal of Accounting Information Systems*, 5:65–81, 2004.
- [OPT05] Alexander Osterwalder, Yves Pigneur, and Christopher L. Tucci. Clarifying Business Models: Origins, Present and Future of the Concept. *Communications of the Association for Information Science (CAIS 2005)*, 15:751–775, 2005.
- [Ora11] Oracle. *MySQL Workbench 5.2*, <http://www.mysql.com/products/workbench>, 2011.
- [Ost04] Alexander Osterwalder. *The Business Model Ontology. A Proposition in a Design Science Approach*. Dissertation, University of Lausanne, 2004.
- [Pac94] Luca Paccioli. *Summa de Arithmetica, Geometrica, Proportioni et Proportionalita (Everything about Arithmetic, Geometry, and Proportion)*. Venice, 1494.
- [Par76] David L. Parnas. On the Design and Development of Program Families. *IEEE Transactions on Software Engineering*, 2(1):193–213, 1976.

- [Pel03] Chris Peltz. Web Services Orchestration and Choreography. *Computer*, 36:46–52, 2003.
- [PMG⁺04] Geert Poels, Ann Maes, Frederik Gailly, Roland Paemeleire, Geert Poels, Ann Maes, and Frederik Gailly. User Comprehension of Accounting Information Structures: An Empirical Test of the REA Model. http://www.feb.ugent.be/nl/Ondz/wp/Papers/wp_04_254.pdf, 2004.
- [Poe03] Geert Poels. Conceptual Modeling of Accounting Information Systems: A Comparative Study of REA and ER Diagrams. In *Proceedings of the 2nd International Workshop on Conceptual Modeling Quality, in conjunction with the 22nd International Conference on Conceptual Modeling (ER 2003), Chicago, Illinois, USA*, pages 152–164. Springer, 2003.
- [Por98] Michael E. Porter. *Competitive Advantage: Creating and Sustaining Superior Performance*. Free Press, 1998.
- [PU03] Günther Pernul and Rainer Unland. *Datenbanken im Unternehmen: Analyse, Modellbildung und Einsatz (in German)*. Lehrbücher Wirtschaftsinformatik. Oldenbourg, 2003.
- [RAM09] Khairina Rosli, Aidi Ahmi, and Liana Mohamad. Resource-Event-Agent (REA) Modelling in Revenue Information System (RiS) Development: Smart Application for Direct-Selling Dealers and SMEs. *Journal for the Avancement of Science & Arts*, 1(1):43–62, 2009.
- [RJB10] James Rumbaugh, Ivar Jacobson, and Grady Booch. *Unified Modeling Language Reference Manual*. Addison-Wesley Professional, 2nd edition, 2010.
- [RM99] Stephen R. Rockwell and William E. McCarthy. REACH: Automated Database Design Integrating First-Order Theories, Reconstructive Expertise, and Implementation Heuristics for Accounting Information Systems. *Intelligent Systems in Accounting, Finance and Management*, 8(3):181–197, 1999.
- [RS11] Marshall B. Romney and Paul J. Steinbart. *Accounting Information Systems*. Pearson Education, Limited, 2011.
- [SA11] Walter Schwaiger and Stefan Achleitner. Integrated ERP Systems. In *Proceedings of the 5th International Workshop on Value Modeling and Business Ontologies (VMBO 2011), Ghent, Belgium*, 2011.
- [SBPM09] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework*. Addison-Wesley, 2009.
- [Sch10] Rainer Schuster. *Requirements Management for B2B Processes: A Worksheet Driven Approach from E3-Value and REA to UMM*. PhD thesis, Vienna University of Technology, 2010.

- [Sed12] Tod Sedbrook. Applying a Business Policy Meta-Model in a Domain Specific Modeling Language. In *Proceedings of the 6th International Workshop on Value Modeling and Business Ontologies (VMBO 2012)*, Vienna, Austria, 2012.
- [SGM02] Julie Smith David, Gregory J. Gerard, and William E. McCarthy. Design Science: An REA Perspective on the Future of AIS. *Information Systems Section Monograph*, American Accounting Association, pages 35–63, 2002.
- [SHH⁺11] Christian Sonnenberg, Christian Huemer, Birgit Hofreiter, Dieter Mayrhofer, and Alessio 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, 2011.
- [SMC07] Spencer Smith, John McCutchan, and Fang Cao. Program Families in Scientific Computing. In *Proceedings of the 7th Domain-Specific Modelling Workshop (DSM 2010)*, in conjunction with the 25th Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2007), Montreal, Quebec, Canada, pages 39–47. ACM, 2007.
- [SMHW10] Rainer Schuster, Thomas Motal, Christian Huemer, and Hannes Werthner. From Economic Drivers to B2B Process Models: A Mapping from REA to UMM. In *Proceedings of the 13th International Conference on Business Information Systems (BIS 2010)*, Berlin, Germany, pages 119–132. Springer, 2010.
- [Sof11] Software AG. *ARIS Platform*, http://www.softwareag.com/corporate/products/aris_platform/default.asp, 2011.
- [SS77] John M. Smith and Diane C. Smith. Database Abstraction: Aggregation and Generalization. *ACM Transactions on Database Systems*, 2(2):105–133, 1977.
- [SZ09] Mark Strembeck and Uwe Zdun. An Approach for the Systematic Development of Domain-Specific Languages. *Software - Practice and Experience (SPE)*, 39(15):1253–1292, 2009.
- [Tho05] Oliver Thomas. Understanding the Term Reference Model in Information System Research. In *Proceedings of Workshop on Business Process Reference Models (BPRM 2005)*, in conjunction with the 3rd International Conference on Business Process Management (BPM 2005), Nancy, France, pages 16–29. Springer, 2005.
- [Tim98] Paul Timmers. Business Models for Electronic Markets. *Electronic Markets*, 8(2):3–8, 1998.
- [TTL00] Don Tapscott, David Ticoll, and Alex Lowy. *Digital Capital: Harnessing the Power of Business Webs*. Harvard Business Scholl Series. Harvard Business School Press, 2000.

- [UN/08] UN/CEFACT. *REA Specialization Module for UN/CEFACT's Modeling Methodology (UMM)*, Public Draft V1.0, 2008.
- [UN/09] UN/CEFACT. *Core Components Technical Specification 3.0 (CCTS)*, http://www.untmg.org/ccts/spec/3_0, 2009.
- [UN/10] UN/CEFACT Technologies and Methodologies Group (TMG). *UN/CEFACT's Modeling Methodology (UMM), UMM Meta Model - Foundation Module - Candidate for V2.0, Draft for Implementation Verification*, http://www.untmg.org/umm/spec/foundation/2_0, 2010.
- [vB07] Jan vom Brocke. Design Principles for Reference Modeling - Reusing Information Models by Means of Aggregation, Specialisation, Instantiation, and Analogy. In *Reference Modeling for Business Systems Analysis*, pages 47–76. Idea, 2007.
- [vDK98] Arie van Deursen and Paul Klint. Little Languages: Little Maintenance? *Journal of Software Maintenance*, 10(2):75–92, 1998.
- [vDKV00] Arie van Deursen, Paul Klint, and Joost Visser. Domain-Specific Languages: An Annotated Bibliography. *ACM SIGPLAN Notices*, 35(6):26–36, 2000.
- [VHHK10] Dominik Vymetal, Frantisek Hunka, Miroslav Hucka, and Josef Kasik. Enterprise Modeling Based on Combination of Process and REA Value Chain Perspective. *Journal of Advanced Research in Management*, 1(2):145–158, 2010.
- [Vos08] Gottfried Vossen. *Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme (in German)*. Oldenbourg, 5th volume edition, 2008.
- [WJA⁺11] Hans Weigand, Paul Johannesson, Birger Andersson, Jeewanie Jayasinghe Arachchige, and Maria Bergholtz. Management Services - A Framework for Design. In *Proceedings of 23rd International Conference on Advanced Information Systems Engineering (CAiSE 2011), London, UK*, pages 582–596. Springer, 2011.
- [WV01] Peter Weill and Michael R. Vitale. *Place to Space: Migrating to eBusiness Models*. Harvard Business School Press, 2001.
- [Yu76] Shih Cheng Yu. *The Structure of Accounting Theory*. The University Presses of Florida, 1976.
- [Zap09] Marco Zapletal. *A UML-Based Methodology for Model-Driven B2B Integration: From Business Values over Business Processes to Deployment Artifacts*. PhD thesis, Vienna University of Technology, 2009.
- [ZJWL10] Guoqiang Zhang, Suling Jia, Qiang Wang, and Qi Liu. REA-based Enterprise Business Domain Ontology Construction. *Journal of Software (JSW 2010)*, 5(5):522–529, 2010.

- [ZLS08] Marco Zapletal, Philipp Liegl, and Rainer Schuster. *UN/CEFACT's Modeling Methodology (UMM) 1.0 - A Guide to UMM and the UMM Add-In*. VDM Verlag Dr. Müller, 2008.



Dipl.-Ing. Mag. Dieter Mayrhofer

Ungerbachstrasse 32
2860 Kirchsschlag
Austria

+43 664 12 10 403

dieter.mayrhofer@gmail.com



„The Best Way to Predict the Future is to Create It.“
(Peter F. Drucker)



Curriculum Vitae

Personal Information

Name Dieter Mayrhofer
Date of Birth September 6th 1982
Citizenship Austria

Education

since 2009 **Vienna University of Technology, Austria, PhD Business Informatics**
B2B integration, Business Models
2006 - 2008 **Vienna University of Technology, Austria, Master program Computer Management**, passed with distinction
2006 - 2007 **Vienna University of Technology, Austria, Master program Medicine and Computer Science**, passed with distinction
2003 - 2005 **Vienna University of Technology, Austria, Bachelor program Medicine and Computer Science**, passed with distinction
1997 - 2002 **Higher Technical College Wiener Neustadt**, passed with distinction (top of the class)

Work Experience

since 2010 **ENTE Startup** - B2B integration/electronic document exchange for small and medium companies and their ERP systems - Wicket, MongoDB, Shiro, Spring, Google Protocol Buffers, CXF, Web Services, Maven, Jenkins
Feb 2008 - Jan 2009 **IBM US, Silicon Valley/Almaden Research Center**
Main developer of a research prototype together with the China team on data-driven XML business intelligence based on DB2-XML, Cube Server, Java, Flex, and Amazon EC2. Prototype is the basis for a product being developed.
Aug 2004 - Aug 2007 **IBM Austria, Vienna, Application Development Team**
Java, JSP, C++, Shell Script, Portlet, DB2 and Websphere App Server
 Aug - Sept 2004 Extending a JAVA e-mail library with Unicode support
 Aug - Sept 2005 Functional web prototype to support collaboration between research and IBM BCS
 Aug - Sept 2006 Automatic feeds and e-mail notification on AIX and Win
 May - Aug 2007 Requirements in back-office area, automatic enrolment into teaching application, Win client application changes
May - July 2005 **Seibersdorf research, Austrian Research Centers**
Video annotation - Tunnel security system
2004 - 2010 **Server/network administrator** for a medical practice
Dec 2002 - May 2003 **Software Dep. Military Academy**, developing JavaScript web calendar
July 2000 **Internship at Medical Computer Ware**, Vienna



International Experience / Achievements / Awards

2012	Workshop chair , local organization of the 6th International Workshop on Value Modeling and Business Ontology, Vienna, Austria
2011	Participant of the TUtheTOP High Potential Program of the Vienna University of Technology (120 students out of 23,000)
2010	PhD Grant received by the faculty of informatics of the Vienna University of Technology
2010	Organizational staff of the 10th International Conference on Web Engineering, Vienna, Austria
Feb 2008 - Jan 2009	Software Engineer, IBM USA, Silicon Valley/Almaden Research
July 2007	Master Class for Medical Informatics 2007 , representing the Technical University at the 6 day International IPHIE in Hall in Tyrol
2000 - 2001	Exchange year Sanger high school CA, USA

Qualifications

Languages	Fluent in German and English
Computer	JAVA, C#, PHP, C/C++, WPF, HTML, XML, JavaScript, AJAX, JSF 2.0, Wicket, Flex, DSL, DB2, SQL, NoSQL, UML, CVS Adobe Photoshop/Premiere, WINDOWS, LINUX, MS Office, NetBeans, Microsoft Visual Studio, Eclipse, Websphere, Portalserver/Applicationserver, Rational Application Developer
Teaching	Teaching in university courses: Web Application Development and Model Engineering, supervisor of master theses for master students
Miscellaneous	Communicative, experience in international software projects, talks in front of international audience, IEEE membership,

Academic Projects

ERPEL	E-Business Registry Permitting Enterprise Liaisons - electronic business document exchange
BSopt	Business Semantics on top of Process Technology - from business models to business process models to deployment artifacts - C#, Domain Specific Languages
Master Thesis	Medical Records on a Tablet PC in Medical Practices - C#, WPF
Class-Scheduler	Web-based class schedule and backup scheduling - PHP, PostgreSQL, JavaScript
Web-Calendar	JavaScript enabled web calendar - PHP, JavaScript, MySQL

References

Prof. Christian Huemer	PhD Thesis Supervisor , Tech. University	+43 1 58801 18882
Qi Jin (Mentor)	TLM Google , former Program Director, STSM at IBM DB2 Warehouse Engine and XMLDevelopment Silicon Valley Lab, USA	+1 408 463 2324
Helmut Riegler	Software Architect , IBM, Austria	+43 664 6185229



Publications

2012

- D. Mayrhofer, 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) held in conjunction with the 24th International Conference on Advanced Information Systems Engineering (CAiSE 2012)", Springer, (2012).
- D. Mayrhofer, C. Huemer:
"Business-Model-Driven Data Engineering Using the REA-DSL"; in: "6th International Workshop on Value Modeling and Business Ontology (VMBO 2012)", (2012).
- B. Hofreiter, C. Huemer, G. Kappel, D. Mayrhofer, J. vom Brocke:
"Inter-organizational Reference Modeling - A Position Statement"; in both: "Business System Management and Engineering in Springer LNCS volume 7350 and Proceedings of the International Workshop on Business System Management and Engineering (BSME2010) @ TOOLS 2010", (2012), 1 - 16.

2011

- C. Grün, C. Huemer, P. Liegl, D. Mayrhofer, T. Motal, R. Schuster, H. Werthner, M. Zapletal:
"eBusiness"; in: "Handbook of Semantic Web Technologies", Springer, Berlin, Heidelberg, 2011, 787 - 848.
- C. Sonnenberg, C. Huemer, B. Hofreiter, D. Mayrhofer, A. Braccini:
"The REA-DSL: A Domain Specific Modeling Language for Business Models"; in:
"Advanced Information Systems Engineering Proceedings of the 23rd International Conference (CAiSE 2011)", Springer, LNCS 6741 (2011), 252 - 266.
- D. Mayrhofer, C. Sonnenberg, B. Hofreiter, C. Huemer:
"A Domain Specific Modeling Language for REA"; in: "5th International Workshop on Value Modeling and Business Ontology (VMBO 2011)", (2011).
- D. Mayrhofer, C. Huemer, B. Hofreiter, C. Sonnenberg:
"REA-XML: An Unambiguous Language for REA Business Models"; in: "The 8th IEEE International Conference on e-Business Engineering (ICEBE 2011)", (2011), 1 - 8.

2010

- D. Mayrhofer:
"Reference Modeling for Inter-organizational Systems"; in: "Proceedings of the MODELS 2010 Doctoral Symposium", ACM/IEEE, (2010), 37 - 42.

2009

- P. Liegl, D. Mayrhofer:
"A Domain Specific Language for UN/CEFACT's Core Components"; in: "Proceedings of the 5th World Congress on Services", IEEE Computer Society, (2009), 123 - 131.

2007

- A. Baierl, E. Derndorfer, T. Vogel-Lahner, D. Mayrhofer, C. Pichler, A. Klostermann, H. Reitner:
"Diet of Students in Rural Areas"; in: "Journal for Nutritional Medicine", 9(1):13 (2007), 123 - 131.