CHAPTER 3

EVALUATION OF EXISTING RDM DESIGN ALTERNATIVES

3.1 INTRODUCTION

In this chapter of this thesis we will give an answer to the following research subquestion that was stated in chapter 1:

*Why do the existing requirements determination approaches from the literature not comply with the quality criteria for assessing requirements determination methods?*

In order to answer this question we will first provide an overview of a number of (families) of approaches that are used in the process of requirements determination and that are documented in the information systems body of literature. Secondly, we will further discuss those approaches that at least contain a modeling language to express the data-oriented perspective of a requirements specification. After we have discussed these ‘families’ of requirements determination approaches we will compare them on the criteria that we have derived in chapter 2. We will analyze a member of each family of approaches and we will discuss the deficiencies of these approaches that need improvement in order to obtain requirements specifications that are precise and consistent, and that fulfill the completeness, domain richness, efficiency and formality criteria. The requirements modeling problems that we will encounter while discussing these approaches will be used when we are going to formulate the operationalized design specification of this thesis in chapter 4.

3.2 A SURVEY OF APPROACHES FOR REQUIREMENTS DETERMINATION FROM THE LITERATURE

Traditionally two ‘families’ of approaches can be distinguished in the field of requirements determination: the *data-oriented* approaches and the *process-oriented* approaches (Bubenko and Wangler, 1992:393). In addition to these two groups of approaches, hybrid approaches have emerged that are both data-oriented and process-oriented (Vessey and Conger, 1994:102). In the mid-eighties the object-oriented approach emerged in which static and dynamic features of an enterprise area should be considered together in objects (Parsons and Wand, 1997:109). In the nineties a business process engineering approach was introduced that provides facilities for the creation of a requirements definition (Scheer, 1998).
3.2.1 Data oriented approaches in requirements determination

The data-oriented approaches to requirements determination can be divided into the following families of semantic data modeling: the ER or Extended ER (Entity-Relationship) approach and the Fact Oriented Modeling approach (NIAM, ORM) (Kim and March, 1995:103). Peckham and Maryanski (1988) reported on a survey of a number of semantic data models. A summary of their findings is given in table 3.1.

Table 3.1 Main findings of Peckham and Maryanski survey (Peckham and Maryanski, 1988:181)

<table>
<thead>
<tr>
<th>Data model</th>
<th>Relationship representation</th>
<th>Derivation/inheritance</th>
<th>Relationship semantics</th>
<th>Dynamic modeling</th>
</tr>
</thead>
<tbody>
<tr>
<td>E-R</td>
<td>Independent and tables</td>
<td>No</td>
<td>User selectable</td>
<td>No</td>
</tr>
<tr>
<td>TAXIS</td>
<td>Entity (classes)</td>
<td>Inheritance</td>
<td>Predefined</td>
<td>Transaction modeling, object oriented</td>
</tr>
<tr>
<td>SDM</td>
<td>Independent and entity (classes)</td>
<td>Elaborate and varied</td>
<td>User defined</td>
<td>No</td>
</tr>
<tr>
<td>Functional</td>
<td>Functions</td>
<td>Functional</td>
<td>User defined</td>
<td>No</td>
</tr>
<tr>
<td>RM/T</td>
<td>Independent</td>
<td>Inheritance</td>
<td>Predefined</td>
<td>No</td>
</tr>
<tr>
<td>SAM</td>
<td>Independent</td>
<td>Summation over classes/ inheritance</td>
<td>Predefined</td>
<td>Object oriented</td>
</tr>
<tr>
<td>Event</td>
<td>Attributes</td>
<td>No</td>
<td>Predefined</td>
<td>Transaction modeling</td>
</tr>
<tr>
<td>SHM+</td>
<td>Attributes, entities, separate</td>
<td>Inheritance over Generalization and Association hierarchies</td>
<td>Predefined</td>
<td>Transaction modeling</td>
</tr>
</tbody>
</table>

In addition to the E-R and fact-oriented approaches, the Semantic Data Model (SDM) by Hammer and McLeod (1981) has had an influence the evolution of the dialects within the ER and Fact-oriented approaches and it has had an influence on the design of object-oriented modeling languages. However, in the survey that we will present in this thesis we will limit ourselves to the ER and Fact-Oriented families of approaches in data modeling.

The Entity-Relationship family of approaches

In 1976 Chen published the first article on Entity Relationship Modeling (Chen, 1976). The basic modeling constructs for capturing the data structure in the ER model are entity(s) (sets), attributes and relationship(s) (sets). The entities are the objects in an application domain about which information is collected; attributes represent intrinsic properties of entities whose value does not depend upon other entities in the model. Relationships represent interconnections among entities (Pitrik, 1996:115). In addition Chen introduced (maximum) cardinalities that can express some of an application area’s business rules. In the course of time Chen’s original Entity-Relationship model has been extended EER (Teorey et al., 1986); ERT (Theodoulidis et al., 1991); EDM...
(Scheer and Hars, 1992); NER (Silva and Carlson, 1995); ER+ (Kolp and Zimanyi, 2000); MEER (Balaban and Shoval, 2002)) in which additional conceptual abstractions have been incorporated (Chan et al., 1998:117): optional relationships (Teorey et al., 1986), subtyping (Teorey et al., 1986), aggregation, generalization/specialization (Gogolla and Hohenstein, 1991; Teorey et al, 1986), time and complex objects (Theodoulidis, 1991), update methods (Balaban and Shoval, 2002). A main characteristic according to Akoka and Comyn-Wattiau of ER models is that these models are mainly oriented towards the data-oriented perspective and leave the process- and behaviour-oriented perspectives undefined (Akoka and Comyn-Wattiau, 1996:88). At the beginning of the 21st century the family of (E)ER approaches remains the most popular approach used in practice (Rauh and Stickel, 1996:135; Shoval and Shiran, 1997:298) and in curricula of universities (in case we consider the static aspects of UML class diagrams as a flavour of the (E)ER family of approaches (Balaban and Shoval, 2002:245; Steimann, 2000:88)). Saiedian surveyed a number of (E)ER models in the literature (Saiedian, 1997) and concluded that in the course of time, the expressiveness of these models has increased and that the recent extensions incorporate object-oriented features, e.g. methods, messages and operations. We will make a distinction into ‘conventional’ EER approaches and ‘object-oriented’ EER approaches. The latter group will be considered object-oriented approaches and is analyzed in section 3.2.3.

The fact oriented family of approaches

From the pioneering work of Abrial (1974) on the Semantic Binary Relationship Model (SBRM), followed by the object-role model (Falkenberg, 1976a, 1976b) the fact-oriented approach became a relatively popular requirements specification approach when the object-role models were expressed in a ‘circle-box’ notation and accompanied by a modeling methodology (ENALIM) (Nijssen, 1977). The ENALIM methodology provided the foundation for (binary) NIAM (Verheijen and van Bekkum, 1982) and the Binary Relationship Model (Van Griethuysen, 1982). In the late 1980’s binary NIAM evolved into N-ary fact oriented information modeling (Halpin and Orlowska, 1992; Leung and Nijssen, 1988; Nijssen and Halpin, 1989) and the acronym NIAM became a shortcut for natural language information analysis method (Halpin, 1996). The data model in the fact-oriented approaches basically consists of a set of fact types and entity types that are connected through roles. A fact type is a semantic relationship consisting of N roles between (at most) N (different) entity types and or label types. Every entity type can be involved in a number of roles within any number of fact types. The content of the fact base at any time is subject to the set of population state and state transition constraints that can be defined on the information structure diagram (ISD, see Verheijen and van Bekkum, (1982)). Prabhakaran and Falkenberg (1988:98) give an overview of other fact oriented modeling approaches, amongst them: CSL (Breutmann et al., 1979); CIAM (Gustafsson et al., 1982); DADES/RM/RA (Olive, 1982), REMORA (Rolland and Richard, 1982).
Conclusions for the data oriented approaches in requirements determination

The data oriented approaches in requirements determination are numerous. The two families of approaches that appear to be the most influential are the entity-relationship family of approaches and the fact-oriented family of approaches (Kim and March, 1995). The commonality of all members of these two families of approaches lies in the perspective of a subject area that they intend to model: the data-oriented perspective, this implies that these approaches have hardly any facilities for modeling the process-oriented and behaviour-oriented perspectives of an application domain.

3.2.2 Process oriented approaches in requirements determination

The process-oriented approaches that we will discuss in this thesis have their origins in the mid-seventies and have in common that they all are based on the notion that application systems can be modeled as a set of functions that interact and that some form of (functional) decomposition is required. We will discuss three (sub) approaches within the process-oriented family: SADT, the structured analysis and structured design (SA/SD) school and ISAC.

SADT

SADT was developed by Douglas T. Ross (Maarssen and McGowan, 1986). A SADT model is a series of data flow diagrams that consists of activities that transform input data onto output data. Control governs the way in which the transformation takes place and mechanisms show the means by which an activity is performed (Maarssen and McGowan, 1986). A transformation can be decomposed into a sub transformation on a lower level of abstraction.

Structured analysis and structured design (SA/SD) school

The structured analysis and structured design school is based upon work of Yourdon and Constantine (1979), Gane and Sarson (1979) and DeMarco (1978). SA/SD looks upon enterprises as functions that process data. The main diagramming technique in SA/SD is the data flow diagram. Data Flow diagrams are designed to show the functionality of an application. A data flow diagram (DFD) consists of a collection of processes, flows, stores, terminators. The processes in a DFD constitute the activities of the application that is being represented. Each process is considered to transform the incoming flows of data and material into outgoing flows of data (and material). Stores are containers for data or material that is carried in the flow. Terminators represent actors or processes external to the system that is under consideration. A transformation can be decomposed into a sub transformation on a lower level of abstraction. Opdahl and Sindre (1994:231-234) point at the omissions in the application of DFD’s for real-world modeling. They conclude that decomposition of DFD’s is problematic with respect to flows. They specifically critique the real world meaning of high-level flows.

Ward (1986) extended the data-flow diagram by concepts that represent control and timing.
ISAC

ISAC (Lundeberg et al., 1979; Lundeberg, 1982) was developed in the 1970’s at Stockholm University and the methodology was created for a much broader coverage of stages in the information systems development life cycle than the requirements determination stage. The ISAC methodology consists of the following stages: change analysis, activity study, information analysis, data system design (Ruys, 1983). Falkenberg et al. (1983:188) concluded that ISAC is very strong in the earliest phases of systems design i.e. change analysis and activity analysis, however the data-model that is the outcome of ISAC is cumbersome and cannot guarantee integrity. Hanani and Shoval (1986:249) conclude that a major gap exists between the products of information analysis and the design of the data system. Furthermore, ISAC does not have facilities for specifying static and dynamic constraints.

Conclusions for the Process oriented approaches in requirements determination

Floyd (1986) compared SADT, the SA/SD school and ISAC and concluded that only SADT and ISAC seemed suitable for requirements determination. Deng and Fuhr (1995:107) claim that structured analysis and design techniques cannot allow a simple modification to a module without a complete redesign of the system.

Henderson-Sellers and Edwards (1990:145) summarize the findings of Meyer regarding the flaws in top-down system design as is implemented in all process-oriented approaches that we have discussed in this chapter:

1. top down systems design does not take account of evolutionary changes,
2. in top down systems design a system is characterized by a single function,
3. top down design neglects the data structure aspect very often,
4. top down design does not encourage reuse.

3.2.3 Object oriented approaches in requirements determination

The object-oriented analysis concepts have their roots in object-oriented programming (Cox, 1986) and object-oriented software construction (Meyer, 1988) blended with ideas from semantic data modeling and knowledge representation (Mylopoulos et al., 1999).

Classic OO approaches in requirements determination

The OO-paradigm has been applied in corporations during the last decade (Johnson and Hardgrave, 1999:5) in methodologies for requirements determination and information systems analysis and design: object-oriented modeling and design (OMT) (Rumbaugh et al., 1991), object-oriented systems design (OOD) (Yourdon, 1994), object-oriented information engineering; analysis, design and implementation (Montgomery, 1994). Misic and Graf (2004) empirically studied the use of different system analysis approaches by systems analysts and concluded that the percentage of respondents that are using object-oriented approaches for analysis and design grew from 0 % in 1994 to 35 % in 2001. Davis (1995) gives a critical view on the application of OO programming concepts for requirements specification and information systems design.
The OO paradigm considers an object as:” an identifiable thing that remembers its own state [1, 6], and that can respond to requests for operations with respect to this state [5].” (Parsons and Wand, 1997:106). Furthermore the OO-paradigm considers an object to have an unchangeable identity (Brown, 1991:20)), it encapsulates data and behaviour and it is persistent (Parsons and Wand, 1997:106). One of the first OO approaches that were specifically designed for requirements determination was OMT (Rumbaugh et al., 1991).

The Unified Modeling Language (UML)

In 1997 a standard emerged (OMG, 2002) to streamline the multitude of OO-approaches: The Unified Modeling Language (UML). An application’s data model can be expressed in an object-oriented class diagram. The static constraints and the static derivation rules can be defined in the static structure part of a class diagram, for example in a UML class diagram this can be done by using association end and attribute multiplicities and the Object Constraint Language (OCL). Furthermore, the dynamic constraints and dynamic rules of a domain application can partly be encoded as methods from object classes and in miscellaneous models that have come into existence like for example use cases and state charts.

Conclusions for the Object oriented approaches in requirements determination

We can conclude that the object-oriented approaches have evolved from the application of the OO-paradigm in programming languages in the seventies and eighties towards OO approaches that are considered suitable for requirements determination. The object-oriented approaches provide facilities for the specification of the data- as well as the process- and behaviour-oriented perspectives in requirements determination.

3.2.4 The Business Process Engineering approach: ARIS

From the late eighties until the mid-nineties the Business Process (Re)engineering was at its peak. Around that time product-software suppliers started to implement their IT solutions on a wide scale in (mainly) large organizations. Scheer has developed an Architectural framework for integrated information systems (ARIS), that can analyze, model and navigate business processes (Scheer, 1994:607). ARIS acknowledges the existence of a semantic requirements definition (Scheer, 1998:14). To express the data view of such a semantic requirements definition, ARIS uses an (E)ER model18. For representing the functional and control view, ARIS uses a number of modeling techniques, like for example flow-chart techniques, Petri-nets, activity-diagrams and OMT object-diagrams, object-flow diagrams.

---

18 ARIS consists of a number of requirements models that are basically covered by the modeling facilities in the other three families of approaches.
3.3 THE SUITABILITY OF EXISTING APPROACH FAMILIES FOR REQUIREMENTS DETERMINATION

In this paragraph we will evaluate the family of approaches that we have introduced in paragraph 3.2 on the first criterion that we have derived in chapter 2. In table 3.2 we have compared these three ‘families’ of approaches with respect to the underlying way of thinking, the way of modeling and their way of working (Wijers, 1991: 17-23).

Table 3.2 Comparison families of approaches found in the literature

<table>
<thead>
<tr>
<th>Way of thinking</th>
<th>Data-oriented</th>
<th>Process-oriented</th>
<th>Object-oriented</th>
<th>Business Process Engineering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Way of thinking</td>
<td>Business data</td>
<td>Functions that interact</td>
<td>Objects that encapsulate data and behaviour</td>
<td>Business processes</td>
</tr>
<tr>
<td>Way of modeling</td>
<td>Information models</td>
<td>DFd’s, A-schemes</td>
<td>Class diagrams, use cases, activity diagrams, State charts</td>
<td>Process chains, OMT class diagrams, ER models, Petri-nets</td>
</tr>
<tr>
<td>Way of working</td>
<td>Analysis of textual description of application domain</td>
<td>Top-down functional decomposition</td>
<td>Identifying objects in application domain</td>
<td>Translating business application knowledge into DP -suitable structures</td>
</tr>
</tbody>
</table>

The completeness criterion that was given in chapter 2, implies the capability of a requirements determination approach to specify at least the data model from the data-oriented perspective. The data model is necessary in order to be able to describe the content of the process- and behaviour-oriented perspectives in a meaningful way. The literature study has revealed that the process-oriented approaches do not provide sufficient modeling constructs that would allow an analyst to create a requirements specification that contains a data model. We will now analyze three families of approaches for requirements determination that have facilities for a data model: The Entity-Relationship approach, the fact-oriented approach and the object-oriented approach (OO). In sections 3.4 through 3.6 we will evaluate a specific member of each of these families (of approaches) on the modeling deficiencies that exist for these approach instances and the remaining criteria that we have given in chapter 2.
3.4 THE EXTENDED (OR ENHANCED) ENTITY-RELATIONSHIP APPROACH

The entity-relationship approach was introduced in a seminal article of Peter Chen (1976). We will consider an ER extension as it can be found in McFadden et al. (1999:85-159) that is a main stream contemporary text book on database management. In addition we will reference those approaches from the ER literature that provide solutions for some of the modeling deficiencies that we will encounter in the McFadden approach.

3.4.1 Deficiencies in the (E)ER way of modeling

In this section we will discuss a number of problems that are related to requirements specifications that use EER as a specification language. These problems are rooted in the definition of the EER modeling constructs and in the ways in which these modeling constructs can be applied in a requirements determination process.

Ambiguities regarding the modeling of N-ary relationships

In most of the examples that are used in the articles, books or instruction manuals that give the definitions of the (E)ER modeling constructs and examples of how these modelling constructs can be applied, no explicit coverage of how to model n-ary (N>2) relationships is provided: ‘Higher degree relationships are possible, but they are rarely encountered in practice, …’ (McFadden et al., 1999:101). Although in some versions of the EER dialect (Connolly et al., 1996: 174-175) a number of examples of N-ary (N>2) relationships are shown, McFadden et al. (1999) do not give illustrative examples of ‘pure’ N-ary relationships, they adapt the ‘pure’ N-ary relationship into either: an associative entity or gerund (McFadden et al., 1999:99-100)\(^{19}\) having one or more relationship attribute or a (N-1) ary relationship having a relationship attribute (McFadden et al., 1999:102). Furthermore, they give the conditions under which a semantic relationship can be encoded as a gerund. However, these conditions, presume knowledge on the cardinality constraints and the properties of the integrated EER schema, and therefore can not be applied to model the initial requirements.

The application of the N-1 relation/attribute modeling construct leads to severe problems whenever the participating ‘concept type’ that must be modeled as an attribute is also involved in other semantic relationships (see the discussion on the instability of EER models further on). Teorey et al. (1986:201) claim that certain relationships of a degree higher than 2 might exist in a UoD and are ‘awkward’ (or incorrect) when represented in a binary form and they explicitly state that: ‘a ternary relationship cannot be reduced to equivalent binary relationships if the relation used to represent it is in 4NF.’ (Teorey et al., 1986:202).

\(^{19}\)“An associative entity (or gerund) is an entity type that associates the instances of one or more entity types and contains attributes that are peculiar to the relationship between those entity instances .” (McFadden et al., 1999:99-100)
No facility for naming conventions of attributes

In the (E)ER dialect in McFadden et al. (1999) there does not exist a modeling provision for the naming convention of some attributes. In figure 3.1a and figure 3.1b it is illustrated how the same piece of domain semantics must either be modeled as the ER diagram in figure 3.1a or as the ER diagram in figure 3.1b. This means that whatever option is chosen, essential domain semantics will be lost.

After analyzing a number of examples in McFadden et al. (1999:105-110) that illustrate the application of the attribute construct. This analysis reveals that a key attribute stands in all cases for a name class; other attributes can actually stand for name classes or concept types. Furthermore, it is not possible to record both the concept type and the name class in the same attribute if that is required (see for example figure 3.1 in which address is a concept type and address_ID is a name class). Most (E)ER dialects do not have guidelines on when to interpret the attribute as a name class or a concept name. A noteworthy exception is the Extended-Entity-Relationship model as defined by Engels et al. (1992) in which data types (that are user-definable) for attributes can be incorporated into the model.

An employee lives on an address
An adress is identified by an address ID

EMPLOYEE
Address

(A)

An employee lives on an address
An adress is identified by an address ID

EMPLOYEE
Address_ID

(B)

An employee lives on an address
An adress is identified by an address ID
At any time an employee should live on at least 2 addresses

EMPLOYEE
Address

(C)

Fig. 3.1 Domain semantics and representation in EER model I
Ambiguous definition of relationship cardinalities for a ternary or higher order relationships

In the EER specification language only a small number of business rules can be modeled as static constraints. Only those business rules that can be expressed as minimum or maximum cardinalities or that can be expressed as the multi-valued qualification of an attribute (see figure 3.1c) can be modeled. The EER approach contains the concepts of minimum cardinality and maximum cardinality. In McFadden et al. (1999:106) the ‘look across, look across’ type of cardinality constraints is used (Dullea et al., 2003) at least for binary relationships. The application of these minimal cardinalities in ‘pure’ N-ary relationships remains unclear. A number of interpretations exist for the cardinalities that refer to a ‘pure’ ternary or higher order relationships (Halpin, 2001a). To avoid this ambiguity all ternary or higher order relationships must be converted into associative entities or gerunds (McFadden et al., 1999: 105) or binary relationships with relationship attributes.

Fig. 3.2 Domain semantics and representation in EER model II (taken from figure 3.17 in McFadden et al., 1999:108)

Each vendor can supply many parts to any number of warehouses, but need not supply any parts.

Each part can be supplied by any number of vendors to more than one warehouse, but each part must be supplied by at least one vendor to a warehouse.

Each warehouse can be supplied with any number of parts from more than one vendor, but each warehouse must be supplied with at least one part.
We can see from figure 3.2 (taken from McFadden et al., 1999;108) that the relationship cardinalities in these situations represent very complicated combinations of business rules. It is not possible to represent the following simple atomic business rules as (straightforward combination) of relationship cardinalities:

\[
\begin{align*}
& A \text{ part must be supplied to at least one warehouse} \\
& A \text{ warehouse must be shipped from at least one vendor}
\end{align*}
\]

McAllister (1998) gives an approach to check consistency of cardinalities in N-ary ER relationships by using cardinality tables. However, the number of cardinality constraints that should be analyzed as a function of the arity (\(N\)) of the relationship will increase exponentially (e.g. from 2 when \(N=2\) to 180 for \(n=5\)).

In an overview article on cardinality constraints in semantic data models, Liddle et al. (1993) give formal definitions for relationship cardinalities in a number of ER dialects (amongst other data models) in which they show that the definition of maximum cardinality in the original Chen’s ER dialect and the Extended ER (EER) by Teorey et al. (1986) are the same (see figure 3.3a and b) and in which the minimum cardinality...
that is added to the original ER model by Teorey et al. denotes the participation status of an entity (instance) from an entity set. In this notational convention a circle that is placed on the connecting line indicates an ‘optional’ participation for the connected entities in this relationship set. For mandatory participation the line that connects the entity type to the relationship set has no special marking. We can conclude that in EER the ‘look here, look across’ type of cardinality constraints is used (Dullea et al., 2003). In the EER dialect that we have analysed in McFadden et al. (1999), the ‘look across, look across’ application of cardinality can only be found in binary relationships, no instances of N-ary (N>2) relationships are given that illustrate the applicability of the minimum cardinality. We refer to Liddle et al. (1993: 239, 246) for the formal semantics of Chen’s ER and Teorey et al.’s XER cardinality constraints.

Instability of EER models because of the existence of the attribute and relationships as information bearing constructs.

The (E)ER approach shows some problems in terms of capturing evolving requirements when a binary relationship is modeled initially as an attribute of an entity type (see figure 3.4). When this modeling decision has been made in the initial stage of a project, this can lead to remodeling when additional domain semantics need to be incorporated into the application’s (E)ER-model (Bots et al., 1990; Halpin, 1996; Storey, 1991:52). In figure 3.1 we have modeled address as an attribute of the entity type EMPLOYEE. If we now want to model the relationship between an address and a zip code we will have a problem because simply adding a relationship will make it impossible to use the relationship cardinalities for modeling the domain semantics that every address needs to have a zip code (see figure 3.4a). In order to be able to model the domain semantics explicitly in the EER model by using cardinalities we need to remodel the original entity/attribute diagram from figure 3.1a into the relationship in the upper part of figure 3.4b. A noteworthy exception in the family of EER dialects is ERT (Theodoulidis et al., 1991) in which it is not possible to model domain knowledge as attributes. In ERT all domain semantics must be encoded as entities and/or as relationships between entities. Lim and Chiang (2000) give an overview on schema-level relationships in (E)ER diagrams.

Incomplete recording of domain semantics when encoded as relationship or attribute.

We remark that in EER it is not possible to exactly denote the sequence in which the name(s) of entity types and the name of the relationship must be read (Halpin, 2001b:315) in order to derive the correct phrasing of the semantic relationship (e.g. address lives at employee or employee lives at address). In case domain semantics are modeled as an attribute of an entity type (see for example figure 3.4 a), the ER approach does not enforce an analyst to record the verbs or predicate of such a semantic relationship. Let us assume that the upper diagram from figure 3.4a represents the following domain semantics:

Employee lives at address
We will now extend our example by a new requirement:

*Employee was born at address*

![Diagram](image)

**Fig. 3.4 Domain semantics and representation in EER model III**

In the best situation we need to remodel the old requirements either by adding the verb on the attribute name or by creating two binary relationships in the adapted model in which the verbs can be explicitly recorded. However, in the latter situation the model will be subject to the ‘verbalization’ problems in ER diagrams that we have discussed earlier. The main problem, however, with the lacking of a verb that is recorded in an ER model, is in the interpretation of the relationship when time has passed, e.g. for application maintenance purposes.

### 3.4.2 Deficiencies in the (E)ER way of working

Most (E)ER family members lack a procedure that exactly specifies how an analyst can derive a semantically correct ER-model in the requirements determination process. The guidelines that Chen (1976) proposes to support the design of an ER schema are not accurate enough to be able to explain how an ER model must be created (Rolland et al., 1995:338). With respect to the static constraints we remark that apart from some participation and cardinality constraints and disjoint constraints associated with super types and subtypes (McFadden et al., 1999:145) the (E)ER approach does not provide
us with modeling facilities to do so. The same holds for the dynamic constraints and the dynamic rules.

In the EER dialect that we have analyzed (McFadden et al., 1999) no explicit procedure is given on how to apply the EER modeling concepts in a requirements determination process. Especially, the condition under which a N-ary semantic relationship must be modeled either as a ‘gerund’ or a ‘pure’ N-ary relationship are missing in the (E)ER dialects that we have studied in our survey. However, there exist some EER dialects in which modeling steps are given. We will now give a summary of three EER modeling procedures: MOODD (Silva and Carlson, 1995), EER (Teorey et al., 1986) and Storey’s EER dialect (Storey, 1991).

Modeling steps in MOODD

In MOODD a rudimentary outline of a requirements determination procedure is given (RSL) that specifies how sentences from a user requirements specification can be translated onto a Nested Entity-Relationship (NER) diagram (Silva and Carlson, 1995:163):

Step (i) Check for synonyms and homonyms
Step (ii) Use a glossary to ensure uniform use of words
Step (iii) Group sentences describing the static properties of the same subjects into O-paragraphs
Step (iv) Group sentences describing the dynamic properties of the same subject into BR (business rules) paragraphs
Step (v) For each O-paragraph, analyze each sentence converting it to the corresponding NER object.
Step (vi) For each BR-paragraph, analyze each sentence converting into the corresponding UPM expression

Teorey’s modeling steps

Teorey et al. (1986) give a logical design methodology for the creation of relational database schemas. The first step of this methodology, however, is directed towards the EER modeling of requirements and consists of the following sub steps:

Step 1.1 Classify entities and attributes
Step 1.2 Identify the generalization hierarchies and subset Hierarchies
Step 1.3 Define Relationships
Step 1.4 Integrate multiple views of entities, attributes, and Relationships

Teorey et al. give guidelines for classifying entities and (multi-valued) attributes but these guidelines assume knowledge of the final schema: “For example, in the above store and city example, if there is some descriptive information such as STATE and POPULATION for cities, then CITY should be classified as an entity. If only CITY-NAME is needed to identify a city, the CITY should be classified as an attribute” (Teorey et al., 1986:204). This means that such a procedure can never be applied in capturing the initial requirements of a domain user because in that stage global knowledge of the schema is not known (see also Bollen, 2002b).
Storey’s modeling steps

In another EER dialect, Storey (1991) gives a procedure that covers not only the creation of a requirements specification in EER but also provides steps that result in the definition of normalized relational tables, that can serve as an input to a DDL of a database implementation. We will summarize the steps from Storey’s procedure that refer to the requirements specification stage in the analysis and design process:

- Step 1: Identify entities
- Step 2: Identify relationships
- Step 3: Check for design problems and eventually go back to step 1

Storey only specifies what potentially must be done in a requirements determination process, e.g. during the first step she advices to make the distinction among entity types, attributes and relationships. However, no explicit rules are given that can guide an analyst in making those modeling decisions in a specific requirements determination process.

3.5 OBJECT-ROLE MODELING (ORM)

3.5.1  Deficiencies in the ORM way of modeling

In section 3.4.1. we have illustrated some of the modeling deficiencies that exist in the EER-approach. A number of these deficiencies have been addressed in the definition of the modeling constructs in Object Role Modeling. The ‘state-of-the-art’ in fact oriented modeling (Halpin, 2001b); however, still has a number of modeling deficiencies that deserve attention.

Semantics of naming conventions in ORM

If we consider the example from figure 3.4 in which we have given a natural language statement of the domain requirements, the naming convention for an address in ORM is the address name (simple reference scheme) or a compound reference scheme that consists of three values: street name, house number and city name (e.g. see the discussion on signification in (Falkenberg, 1976a)). However, we have assumed that in this UoD, the addresses are restricted to one country. In case a postal service organization decides to expand its activities by taking over a foreign postal service it becomes clear that what used to be a valid signification within the country of origin now has become an invalid or incomplete signification. To avoid these problems from happening when requirements evolve it is a good practice to model these explicit semantics of naming conventions in the requirements specification at all times. An example of an explicit naming convention for an address within the Netherlands would be:
An address within the Netherlands can be identified among the union of addresses within the Netherlands by the combination of street name, house number and city name.

The existence of different referencing modes

In ORM three different ways of modeling naming conventions or reference schemes exist (Bollen, 2002b). The 1-1 referencing mode is depicted graphically by adding the name of the reference mode in parentheses to the name of the entity type that has to be referenced (Halpin, 2001b:81). We have given the ORM model for the following domain requirements in figure 3.5:

```
An employee works for a department within the ABC company
An employee can be identified by an employee ID
A department can be identified by a department name
An employee can work for one department at most
```

![Fig. 3.5 Domain semantics and representation in ORM model I](image)

The second way of modeling naming conventions is the case of a compound referencing scheme in which an entity of a given entity type can only be identified when using 2 or more values. In figure 3.6 we have illustrated such a compound referencing scheme (Halpin, 2001b: 192-195) for our (running) example from figures 3.2 and 3.5 in which we have changed domain semantics that allow us to identify an employee by the combination of first name and last name.

```
An employee works for a department within the ABC company
An employee has exactly one first name
An employee has exactly one last name
An employee can be identified by a combination of first name and last name
A department can be identified by a department name
An employee can work for one department at most
```

If we compare the ORM models from figures 3.5 and 3.6 we can see that the distinction between a simple and a compound reference scheme has a big impact on the resulting Object-Role Model. All other domain semantics in the example of figures 3.5 and 3.6 are identical, however, in the example from figure 3.6 we have a model that contains 3 fact types in comparison with the model from figure 3.5 in which we have only one fact type.
The third way of modeling naming conventions is called objectification (Halpin, 2001b:85) in which a nested object type is modeled as a fact type in which the constituting entity types and/or name types of the objectification are given (see figure 3.7).

An employee works for a department within the ABC company
An employee can be identified by a combination of first name and last name
A department can be identified by a department name
An employee can work for one department at most

The resulting ORM diagram is given in figure 3.7.

In ORM we are not required to specify a role name every time a fact type is defined. This can lead to confusing situations in case the same entity type plays two or more...
roles within a single fact type. Consider, for example, the following application domain semantics.

A Person introduces a person to a person
A person can be identified by a person name
A person can only be introduced once to another person

The resulting ORM diagram is given in figure 3.8.

![ORM Diagram](image)

**Fig. 3.8** Domain semantics and representation in ORM model IV

As we already discussed in chapter 1 of this thesis, we consider the notation legend of a requirements specification of minor importance in comparison with the modeling language concepts. If we now consider the ORM example in figure 3.8 we miss the naming conventions for the major modeling concepts in ORM: fact types and roles that would allow us to communicate the modeling results without having to use a specific notational convention, e.g. we must be able to record the modeling results in figure 3.8 in the following way:

There is a fact type that contains roles person1, person2 and person3.
Fact type template of this fact type reads as follows: <Person1>
introduces <person2> to <person3>.
Role 'person1' is played by the entity type 'Person'.
Role 'person2' is played by the entity type 'Person'.
Role 'person3' is played by the entity type 'Person'.
The name class 'Person name' is a reference type for the entity type 'Person'.
There is a uniqueness constraint defined on roles 'Person2' and 'Person3'.

This means that ORM at least a simple naming convention should exist that will allow an analyst to uniquely identify a role among the union of roles or a compound reference scheme in which a role can be identified by a combination of a fact type name and the (locally unique) role name.
3.5.2 Deficiencies in the ORM way of working

With respect to the availability of a modeling procedure that guides an analyst in creating semantically correct ORM models we remark that ORM has a conceptual schema design procedure (Halpin, 2001b; Halpin and Orlowska, 1992).

Halpin’s conceptual schema design procedure

In ORM a conceptual schema design procedure is defined (Halpin and Orlowska, 1992; Halpin, 2001b). This procedure consists of 7 steps:

- Step 1: From examples to elementary facts
- Step 2: Draw fact types and populate
- Step 3: Trim schema; Note basic derivations
- Step 4: Uniqueness constraints, arity check
- Step 5: Mandatory roles and logical derivation check
- Step 6: Value, Set and Subtype Constraints
- Step 7: Other constraints; Final checks

However, a close examination of this procedure in Halpin (2001b) and Halpin and Orlowska (1992) reveals that the procedure basically tells an analyst what to do next but does not exactly specify how such an activity must be carried out in a requirements determination process. With respect to steps 4, 5, 6 and 7 we must remark that ORM does not give a precise algorithm or procedure the application of which guarantees that the instances of those static and dynamic constraint types will be found in the requirements determination process.

3.6 THE UNIFIED MODELING LANGUAGE (UML)

The Unified Modeling language has its ancestors in a number of object-oriented modeling approaches (OMT (Rumbaugh et al.,1991); OOAD (Booch, 1994); OOSE (Jacobson et al., 1992)). The UML started out as a collaboration between the designers of the latter OO-methods (Kobryn, 1999:30). The UML is “a general-purpose visual modeling language that is used to specify, visualize, construct, and document the artifacts of a software system…it is intended for use with all development methods, lifecycle stages, application domains, and media.” (Rumbaugh et al., 1999:3). In UML the class diagram represents the data-oriented perspective of an application domain (Otero and Dolado, 2004). Bollen (2002c) has analyzed the diagrams types within UML that jointly cover the description of the information requirements as given in the criteria from chapter 2, and has found modeling problems that occur in the application of UML. He concludes that out of the 9 diagram types that are currently defined in UML (class diagrams, object diagrams, use-case diagrams, sequence diagrams, collaboration diagrams, state charts, activity diagrams, component diagrams and deployment diagrams) only the use-case diagram, class diagram, activity diagram and (advanced) state chart diagram are necessary to fulfill the completeness criterion in
section 2.1 of this thesis. Otero and Dolado (2004) conclude that 4 types of diagrams are needed to specify the behaviour-oriented aspects of systems: sequence, collaboration, state and activity diagrams. Dori (2002:83) claims that “The tight interdependence of structure and behavior mandates that these two major system aspects be addressed concurrently. This task is, however, counter-intuitive and extremely difficult if structure and behavior are forced into two (let alone nine) separate diagram types.”

Related to the problem of too many diagram types is the lack of consistency when it comes to modeling for example a state transition constraint as a state chart that constrains the states of the object that are specified in an object class diagram.

![Fig. 3.9 Lack of coherence in UML class diagram and UML state chart diagram](image)

From figure 3.9 we see that the right-hand diagram is intended to serve as a way to encode a transition rule or dynamic constraint on the subsequent values of the attribute marital status of the object class Person. UML does not give guidelines how to consistently model that the state in the state chart refer to a particular attribute of the accompanying class diagram.

In section 3.6.1 we will focus on the deficiencies in the UML specification language that are connected to the modeling constructs for the UML class diagrams

### 3.6.1 Deficiencies in the UML way of modeling

Although the static aspects of class diagrams share most of the modeling problems that were encountered when we analyzed the (E)ER modeling approach, UML has addressed some of them. For example, the naming conventions for attributes are implemented in the UML class diagrams as attribute types. However, there are additional modeling complications that must be taken into account when evaluating the modeling constructs in UML class diagrams and that can be fully contributed to the properties of the object-oriented paradigm, most notably the object ID and the interaction between the concepts of generalization/specialization and class inheritance.

*The object ID in the Unified Modeling Language*

In addition to the declaration of the object’s class, the declaration of the attributes and methods that an object inherits, the OO paradigm states that each object instance has a
‘unique’ identity: “Each object has its own unique identity. Most object-oriented languages automatically generate implicit identifiers with which to reference objects” (Rumbaugh et al., 1991:24). “Object identifiers must uniquely identify as many objects as may ever coexist in the system at any one time” (Cox, 1986:54).

In UML the following definition for the object ID is given: “Each object has its own unique identity and may be referenced by a unique handle that identifies it and provides access to it.” (Rumbaugh et al., 1999:360). This concept of ‘globally’ unique object ID’s to identify objects within a specific application system allows us to make a precise distinction between two different objects that have the same state and behaviour (Dittrich, 1990:16). The existence of these object IDs allows us to refer to a house inhabitant with object ID 234 having the name Tommy and a house inhabitant with object ID 235 having the name Tommy as two different objects (see figure 3.10). It is impossible to empower users in the application domain to use ‘abstract’ object IDs as naming conventions (Halpin, 2001b:353). The best way to encode a domain-based naming convention for the concepts that are modeled as object classes is as a combination of class attributes. UML, however, does not provide a standard graphic notation for such a constraint. Halpin and Bloesch (1999:12) define a primary identifier constraint (’{P}’) on the combination of attributes that can be used to identify an instance of an object class using application-based naming conventions. This means that in UML state constraints need to be applied in order to facilitate the implementation of domain-based naming conventions.

The interaction between the concepts of generalization/specialization and class inheritance.

The OO paradigm uses the same ‘is-a’ relationship for denoting specialization and generalization. In the OO-paradigm: “Generalization and specialization are two different viewpoints of the same relationship, viewed from the super class or from the subclasses. The word generalization derives from the fact that the super class generalizes the subclasses. Specialization refers to the fact that the subclasses refine or specialize the super class.” (Rumbaugh et al., 1991:42). The concept that is used in the OO paradigm for modeling generalizations is the abstract class concept in combination with the ‘is-a’ relationship.
The abstract class can be extensionally defined as the union of extensions of the subclasses at any point in time.

\[
\text{Person} := \text{Tennis player} \cup \text{Employee}
\]

One of the significant concepts in the object-oriented paradigm is the concept of inheritance. Rumbaugh et al. (1991:42) give the following description of inheritance: “…inheritance refers to the mechanism of sharing attributes and operations using the generalization relationship.” Other definitions found in the literature are: “Inheritance is a code-sharing mechanism. It allows reuse of behaviour of a class in the definition of new classes. Subclasses of a class inherit the data structure and the operations of their parent class (also called a super class) and may add new operations and new instance variables.” (Tkach and Puttick, 1994:21). “Inheritance is a tool for organizing, building and using reusable classes” (Cox, 1986:69). For an in-depth discussion on different types of inheritance see Rahayu et al. (2000).

In this case the class hierarchy is determined by clustering characteristics of the class attributes and methods. “Although many of the classes do not represent physical objects, they are conceptual entities which can be stated in the terminology of the problem domain.” (Korson and McGregor, 1990:46). “The availability of an inheritance relation enables the designer to “push higher” and to identify commonality among abstractions and to produce higher level abstractions, from this commonality.” (Korson and McGregor, 1990:53). Bollen (2002d) gives an example of how the aforementioned ‘pushing higher’ process interferes with the specialization/generalization concepts in the data perspective.

The application of the OO concept of inheritance can lead to the creation of abstractions in an object class hierarchy that do not represent things, entities or concepts in a Universe of Discourse. This type of abstraction should be modeled as an abstract object class or the conditions under which it can be modeled as a non-abstract object class should be explicitly given in a methodology for the OO-modeler. Snoeck and Dedene (1996:179-180) offer some guidelines for specializations/generalizations in object-oriented conceptual modeling.
Association end multiplicities in N-ary relationships in UML class diagrams

As we indicated earlier the static aspects of the UML class diagrams are based to a large extent on the (Extended) Entity-Relationship model. However, when the association (end) cardinalities for ternary (or N-ary in general) relationships are discussed, the defining UML literature gives specific definitions. On page 61 of the UML notation guide (Rumbaugh et al., 1999) we find the following definition of association end multiplicity: “Multiplicity for N-ary associations may be specified but is less obvious than binary multiplicity. The multiplicity on a role represents the potential number of instance tuples in the association when the other N-1 values are fixed.” On page 348 of the UML language reference manual (Booch et al., 1999) we find the following definition of association end multiplicity: “In a n-ary association, the multiplicity is defined with respect to the other n-1 ends. For example, given a ternary association among classes (A, B, C) then the multiplicity of the C end states how many C objects may appear in association with a particular pair of A and B objects. If the multiplicity of this association is (many, many, one), then for each possible (A, B) pair, there is a unique value of C. For a given (B, C) pair, there may be many A values, however, and many values of A, B and C may participate in the association.” “If the multiplicity of this association is (many, many, one)……… For a given (B, C) pair, there may be many A values, however.” This means that the upper multiplicity of many (*) defined on the association end that is connected to object class A implies there can exist many links in the object diagram for every possible (B, C) pair. It is not clear whether a lower or implied lower multiplicity for 0 in a n-ary association in UML specifies whether an object in the object class that is connected to the association end can exist independently of the association or not. In Bollen (2002a) an example is given of the ambiguity for the definition of the lower association end multiplicity in UML. Because of this ambiguity or ‘fuzziness’ in the definition of the lower association end multiplicity the expressiveness of this type of graphical constraint type in UML is rendered insignificant.

Verbalization of sentences for N-ary associations

The precise verbalization of the semantics of a n-ary association in UML is not possible (Halpin and Bloesch, 1998).

Default existence of object classes

In UML the modeling of semantic relationships as associations between object classes implies that instances of these object classes can exist on their own (Bollen 2002b), UML does not give guidance on how to suppress these non-existing domain semantics.

Association class and qualifier as naming conventions

Next to the identification attribute(s) that can be used to model the domain naming conventions in UML, there exist a number of alternative referencing modes: the association class and the association qualifier (Bollen, 2002b). However, the choice of a specific referencing mode can only be justified if additional domain semantics have been analyzed.
3.6.2 Deficiencies in the UML way of working

UML is a modeling language without a modeling process or procedure (Liang, 2003:83). Liang (2003) gives a procedure for mapping use cases into classes of a class diagram. In UML a requirements determination procedure is lacking (Bollen, 2002c) that specifies how the UML can be used to model the domain requirements in terms of the data model, static constraints, dynamic constraints, static derivation rules and dynamic rules. Bollen (2002c:24) proposes an outline of a modeling procedure that can be used for applying the necessary UML modeling concepts in order to model those domain semantics that are necessary for the requirements determination (see section 2.1). In the defining UML literature (Booch et al., 1999; OMG, 2002; Rumbaugh et al., 1999), however, such a rudimentary procedure outline is missing and therefore, the consistent application of UML modeling constructs can never be guaranteed.

Juristo et al. (1999:140) give an overview of research that indicates that there are no rigorous criteria for identifying the components of OO conceptual models other than procedures that contain steps that tell an analyst what to do, instead of how. See for an example Nanduri and Rugaber (1996) who took one of the predecessors to UML as their OO methodology: OMT (see Rumbaugh et al., 1991).

Rumbaugh’s modeling steps

The modeling procedure that is recommended by Rumbaugh et al. (1991) and summarized in Nanduri and Rugaber (1996:10) contains the following steps:

- Step 1: Identify objects and classes (nouns)
- Step 2: Identify associations between objects (verb phrases)
- Step 3: Identify attributes of objects and associations (adjectives)
- Step 4: Identify operations (verbs and adjectives)
- Step 5: Organize and simplify object classes using inheritance
- Step 6: Iterate and refine the model

3.7 CONCLUSIONS ON THE WAY OF -MODELING, -WORKING AND -CONTROLLING FOR THE REQUIREMENTS DETERMINATION APPROACHES FROM THE LITERATURE

We now have analyzed three members of the three most prominent families (ER, fact orientation, object orientation) of requirements approaches from the literature. While analyzing specific instances of these three approaches (McFadden’s EER, ORM and UML) we have discovered modeling deficiencies in each of them. Although a number of deficiencies that we, for example, have found in approach A might have been addressed in approach B, the conclusion so far is that each of these three approaches contains some deficiencies. In this chapter we will therefore summarize the extent to which any deficiency that is found in a single approach is addressed or is not addressed in at least one of the other methodologies. In addition we will indicate the extent in
which the three specific modeling approaches comply with the other criteria for requirements determination methods that we have given in chapter 2 of this thesis.

3.7.1 Overall modeling deficiencies

The (E)ER and ORM approaches basically allow an analyst to incorporate all application semantics (static and dynamic if applicable) that can be modeled by the approach into one diagram type e.g. an ER-schema or an ORM information model or information grammar. In UML there exist a multitude of diagram types in which it remains unclear what diagram types must be used for the modeling of the application system’s dynamic features (Dori, 2002). However, for the main purpose of the research in this thesis we have already stated that the notations that are used by the different RDM’s are of secondary importance.

3.7.2 Modeling deficiencies regarding the data model for the way of modeling

In this section we will compare the deficiencies as we have found them in the three approaches which mainly are concerned with the data model, e.g. the definition and naming of domain concepts and their semantic relationships.

Modeling facilities for n-ary relationships

ORM provides modeling support for N-ary and binary relationships. In ORM a binary relationship is a special case of a N-ary. The definitions of uniqueness and mandatory role constraints are orthogonal to the arity of the fact type(s) in the information structure diagram in ORM (this means that ‘look here, look here’ variant is applied for all arities).

It is possible, however, to model N-ary relationships in the EER approach and the UML class diagrams. However, only a few EER dialects, explicitly point at the necessity of a N-ary relationship concept (e.g. Teorey et al., 1986: 202; Thalheim, 2000:40). The main difference between the modeling facilities for N-ary relationship in EER and UML on one side and ORM on the other is in the dependency that exist between the application of cardinality constraints (or association end multiplicities) and the modeling of relationships/associations because some common business rules can not be expressed easily at all times in EER and UML because of ambiguities in the definition of participation cardinalities.

The existence of multiple information bearing constructs

In (E)ER and UML at least two information bearing constructs are available, in EER these are the attribute and relationship, in UML this is the class attribute and association.

In ORM the fact type is the only information bearing construct. An exception within the plethora of EER/OO approaches for requirements determination approaches in terms of the number of information bearing constructs, is Embley’s et al. OSA (Object oriented Systems Analysis) approach (Embley et al., 1992) in which the
declarative information is represented in the Object Relationship Model. In this model
the single information bearing construct is the relationship.

Facilities to capture precise domain semantics of naming conventions

UML and ORM provide facilities for capturing (at least) the names of the name classes.
Most (E)ER dialects lack a facility for recording name classes for concepts
that are modeled as attributes at all times. All three approaches lack a way of explicitly
recording the context in which the names of a name class are valid for referencing
entities or concepts of a given type.

The co-existence of different referencing modes including object ID’s

In ORM three ways of referencing entities exist and in UML entities can be referenced
using a combination of attribute (names)\textsuperscript{20} or as an association qualifier or as an
association class.

In EER entities that need to be referenced by names for a name class can only
be modeled as entity types in which attributes or composite attributes can be applied
(McFadden et al., 1999: 219). In most EER dialects gerunds can be defined which is
similar to association class construct in UML.

Facilities for specification of how to communicate semantic relationships in data
models

In EER and UML there’s no facility for verbalizing the relationships that are modeled
as N-ary relationships in a precise and unambiguous way. In UML verbalization into
sentences is only possible for binaries associations that use an optional marker (Halpin
and Bloesch, 1999:8). In ORM these facilities exist for all semantic relationships in an
application domain.

Naming conventions for elements/concepts in data models

If we want to communicate the content of the data model in a way that is ‘diagram-
free’ we miss naming conventions for the roles and/or fact types in ORM. Furthermore,
the optionality of the role concept in EER and UML can lead to additional application
model verbalization problems. This severely impacts the traceability of the requirements
documents in ORM, EER and UML.

The facility to capture the precise generalization/specialization semantics.

In most extended ER approaches (Balaban and Shoval, 2002; Kolp and Zimanyi, 2000;
Scheer and Hars, 1992; Silva and Carlson, 1995; Teorey et al. 1986; Theodoulidis et
al., 1991) and ORM (Halpin, 2001b) modeling constructs are defined that enable the

\textsuperscript{20} Such a combination of attribute names, however, needs a static constraint that specifies that
the\textsuperscript{20} attribute\textsuperscript{20}s can serve as a reference type. This means that in UML, domain based
naming conventions are encoded as static constraints.
analyst to model the specialization/generalizations relationships that exist in the application domain.

In UML, however, it is possible to create ‘inheritance’ trees in which the generalizability of methods determines a specialization/generalization hierarchy other than is justified by the domain ontology.

3.7.3 Modeling deficiencies regarding the static constraints for the way of modeling

Extent in which business rules can be modeled as static constraints

The business rules that can be modeled as static constraints in EER reflect those domain semantics that can be encoded as cardinalities in binary relationships. The ER+ dialect in addition contains a subset constraint (Kolp and Zimanyi, 2000), Rochfeld and Negros (1992) define a range of inter-relationship constraints in their ER dialect; inclusive FIC, exclusive FIC, simultaneity constraint.

In UML this is extended to include attribute multiplicities. Furthermore UML has the facility to model some types of exclusion and subset constraints. Furthermore UML has the object constraint language (OCL) that enables it to model a wide range of domain semantics.

ORM offers the most pre-defined graphical static constraint types for encoding business rules.

Interpretation of cardinality constraints/association end multiplicities

The minimum relationship cardinalities and/or association end multiplicities in many EER dialects and in UML, especially for N>2 are not or at best ill-defined. In EER a number of interpretations exist for cardinality constraints. Dullae et al. (2003) give two archetypes of interpretations (‘look here, look across’(LELA) and ‘look across, look across (LALA)). In the EER flavor that we have analyzed in this thesis (McFadden et al, 1999: 85-165) we have the LALA variety for binary relationships. However for N-ary relationships the cardinality semantics totally change. McFadden et al. use two ways for encoding N-ary domain semantics: a N-ary relationship as an associative entity of arity (<= N) having 1 or more relationship attributes or as a binary having at least 1 relationship attribute. However, for those application areas in which it is not possible to identify a concept in the application domain as gerund and in which it is not possible to use relationship attributes, the interpretation of the minimum cardinalities for such a ‘pure’ N-ary (EER) relationship remains ambiguous and the fact that the ER approach is used for the creation of requirements specifications, does not give any guidance in how to interpret cardinalities (see figure 3.3).

The multiplicity constraints on association ends defined in UML specify any range of occurrence frequencies applied to a single role for binaries (for N-aries, such a range indicates what occurrence numbers are possible when the other n-1 classes have a fixed value). ORM partitions this multiplicity concept into the orthogonal constraint types: mandatory role constraints and frequency constraints (Halpin and Bloesch, 1999:11).
Default existence constraints

In UML entities or objects are allowed to ‘exist’ independently of the relationships they are involved in (Bollen, 2002b:133).

In EER entity types are strong by default (McFadden et al., 1999: 92-93) which means that they are allowed to exist independently of the relationships they are involved in.

In ORM entity types are not allowed to exist independently by default. Bollen (2002b) concludes that in UML and for the same reason in EER when a (binary or higher order) semantic relationship is modeled, unary relationships that declare the existence of entities or objects are modeled at the same time. This means in practice that to be able to model such a (binary or higher order) relationship (on its own) the analyst has to declare in EER that the entity type that is not allowed to exist independently is assigned the status weak (Kolp and Zimani, 2000: 1059; Tsichritzis and Lochovsky, 1982:182) and in UML a textual constraint must be attached that states that each instance should at least participate in one of the relationships (Bollen, 2002b:133). We note that in an evolving requirements specification this implies that such a constraint must again be specified whenever a new relationship in which the object class participates is added to the EER diagram or UML class diagram.

3.7.4 Modeling deficiencies regarding the dynamic constraints for the way of modeling

In this paragraph we will compare the EER model, the UML and ORM on the facilities that they provide for modeling dynamic constraints. We will use a number of subclasses of dynamic constraints that can be found in De Brock (2000). De Brock makes a distinction into subclasses of dynamic constraints. Prabhakaran and Falkenberg (1988) give modelling constructs for transition oriented constraints (TOC) in NIAM.

Cumulativity of tuples, key attribute value combinations, attribute value combinations

In the terminology of the application information base this cumulativity requirement expresses that every fact that has been entered into the application’s information base should stay in the application’s information base. In EER no provision for such a domain rule exist, in UML the changeability qualification can be defined on an attribute or association end of binary associations (Halpin, 2001b:393) and be assigned the value add Only (Rumbaugh et al., 1999: 166, 184). In ORM changeability constraints are not supported (Halpin, 2001b:395).

Non-decreasing attribute values and non-decreasing number of tuples

These constraint types can not be specified in EER and ORM.

Integrity constraints on initial values

UML may assign initial values to attributes, EER and ORM (Halpin, 2001b:390) do not support this.
Life cycles

UML supports this in the form of state charts and ORM uses a state transition fact type in which the graphs in the life cycle can be captured as data (Halpin, 2001b:299). EER does not support this type of constraint.

Changing Life cycles

UML supports this in the form of state charts, but a change in life cycle implies remodeling. ORM uses a state transition fact type in which the graphs in the life cycle can be captured as data (Halpin, 2001b:299) and therefore changes in the life cycle can be implemented on an information base level. EER does not support this type of constraint.

3.7.5 Modeling deficiencies regarding the static derivation (rules) for the way of modeling

In the specific EER dialect that we have studied (McFadden et al., 1999) only provisions are given for static derivation (rules) that refer to derived attributes. Furthermore these derived attributes are restricted to those that can be derived from other attributes (McFadden et al., 1999:95). It remains unclear whether derived attributes that partly need relationships instances as an input should be signified. In most cases, however, no modeling constructs in EER are given that allow us to model a precise specification of a derivation rule. Rauh and Stickel (1996) give an extension to the ER approach called ERMaed, which contains modeling, constructs for derivation rules.

In ORM, derivation rules are written as text below the diagram (Halpin, 2001b, 97). We note that derivation rules should contain explicit references to roles in the information structure diagram. We note however that the data structure of a derived fact type is not required to be contained in the diagram (Halpin, 2001b:99) but if it is, it must be distinguished from the base diagram by an asterix (Halpin, 2001b:100).

In UML a static derivation (rule) is modeled as a derived element (i.e. a derived attribute or a derived association) (Rumbaugh et al., 1999:254-255). We note that in UML, the derived attribute or association is included in the class diagram and the derivation rule is specified and included in the class diagram. Furthermore, UML allows us to specify (the more complicated) derivation rules (in terms of the number of classes and relationships involved in an activity diagram (Bollen, 2002c:23))

3.7.6 Modeling deficiencies regarding the dynamic rules for the way of modeling

The EER dialect in McFadden et al. (1999) does not provide facilities for the modeling of dynamic rules. Gorman and Choobineh (1991) and Silva and Carlson (1995) do provide an object-oriented extension to ER that facilitates the modeling of dynamic rules.

UML provides modeling facilities for the encoding of event-condition action constraints. Bollen (2002c) states that an advanced state chart in UML can be used to model the event-condition action constraints (Rumbaugh et al., 1999: 447-448).

3.7.7 Modeling deficiencies regarding the way of working and way of controlling

In the EER and OO families of requirements modeling languages, some authors have tried to define a modeling procedure; however these procedures basically specify what an analyst should do rather than prescribing how these steps must be performed. In combination with the choices that are inherent to the multitude of information bearing constructs in EER and UML, these procedures are prone to a ‘deadly embrace’ in terms of the knowledge on the end result that must be available before the initial requirements can be modeled.

Another deficiency in many EER dialects is that the requirements specification that is expressed in such an EER diagram is not complete in terms of domain semantics. In some approaches subsequent steps are given that should transform the requirements specification into an implementation schema, e.g. a relational schema. This, however, means that in this transformational stage from a requirement specification into a design specification domain, still domain knowledge needs to be ‘injected’ to determine the appropriate functional dependencies (see Teorey et al. (1986) and Ram (1995)).

In ORM all semantics regarding functional dependencies are incorporated in the information model or conceptual schema. In the fact-oriented approach, the conceptual schema design procedures in Halpin and Orlowska (1992) and Halpin (2001b), however, do not specify how the instances of the pre-defined constraint types can be instantiated at all times.

3.7.8 Summary of Modeling deficiencies in the EER, ORM and UML approaches

In table 3.3 we have summarized the deficiencies from the three approaches studied. A ‘+’ denotes that an approach does not have this language or procedure deficiency. A ‘0’ means that an approach has this deficiency to some extent. A ‘- ‘ means that an approach has this deficiency to the highest extent.
Table 3.3 Summary of the comparison of EER, ORM and UML approaches on modeling deficiencies.

<table>
<thead>
<tr>
<th>Modeling deficiencies</th>
<th>EER</th>
<th>ORM</th>
<th>UML</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQUIREMENTS LANGUAGE/PROC. Deficiency</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Model</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Facilities for n-ary relationships</td>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Facilities for sem. of naming conventions</td>
<td>0</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Existence of multiple information bearing constructs</td>
<td>0</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Facilities for naming conventions of modeling concepts</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Co-existence of different reference modes</td>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Facilities for capturing verbs in data models</td>
<td>-</td>
<td>+</td>
<td>0</td>
</tr>
<tr>
<td>The facility to capture the precise generalization/specialization semantics</td>
<td>-</td>
<td>+</td>
<td>0</td>
</tr>
<tr>
<td>Static constraints</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extent in which static constraints can be modeled</td>
<td>0</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Interpretation of cardinalities/sem’s</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Default existence constraints</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Dynamic constraints</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cumulative of value combinations</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Non-decreasing values</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Integrity constraints on initial values</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Life cycles</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Changing life cycles</td>
<td>-</td>
<td>+</td>
<td>0</td>
</tr>
<tr>
<td>Static Derivation</td>
<td>0</td>
<td>+</td>
<td>0</td>
</tr>
<tr>
<td>Dynamic Rules</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
</tbody>
</table>

3.8 THE SUITABILITY OF EXISTING APPROACHES FOR REQUIREMENTS DETERMINATION WITH RESPECT TO THE COMPLETENESS-, DOMAIN RICHNESS, EFFICIENCY AND FORMALITY CRITERIA

In chapter 2 we have defined four criteria that can be considered relevant in the context of requirements determination. In this section we will summarize the findings from the literature survey on existing requirements determination approaches with respect to these four criteria: domain richness, completeness, efficiency, and formality.

With respect to the domain richness criterion we remark that this criterion contains a number of dimensions. The dimension perception refers to the extent in which different domain users have a different perception of an underlying reality. This means that the application of a requirements determination method must lead to a requirements specification that reflects the (possibly) different perceptions of an underlying reality by different user groups. It is possible to reflect these difference perceptions by using the EER, UML and ORM approaches, whenever they are embedded in a procedure that enables an analyst to integrate the different views from
different user groups on the ‘underlying reality’ by integrating the sub-schemas of these users into a final ‘overall’ requirements specification in which the different perceptions are made explicit. So far, the EER, UML an ORM approaches that we have discussed in this chapter do not give provisions for this, however, Hayne and Ram (1995:100-101) report on a design checking tools (EasyER, GAMBIT and DDEW) for (E)ER models.

The dimension turbulence characterizes the extent in which an application domain is subject to changes in the business data and business rules. While discussing the characteristics of the data model and the static constraints in the EER, UML and ORM approaches we have remarked that there is an interaction between the definition of set of modeling constructs and the extent in which a specification has to be remodeled when requirements are added to the model or change in general. We concluded that the EER and UML approaches are most prone to remodeling because of the multitude of information bearing constructs (Halpin and Bloesch, 1999:8). ORM addresses those issues mentioned but has a problem with a multitude of naming conventions which might lead to unstable models.

With respect to the dimension tacitness, we can say that the EER, UML and ORM approaches basically have the assumption that users will be able to express their initial requirements in natural language, e.g. in a way that the data model, (static and dynamic) constraints and static derivation and dynamic rules can be written down in a requirements document. This restricts the applicability of these approaches to those domains that exclusively contain explicit knowledge. However, we think that a requirements determination methodology must be able to capture (at least some of) those tacit business rules that are implicit but that can be made explicit in the terminology of Kim et al. (2003).

With respect to the dimension anchoring we can say that the requirements determination process in which we use EER and UML models for our specification language are in principle not limited to any specific range on the anchoring scale. ORM is anchored in familiar examples or data use cases (Halpin, 2001b:60) and it requires the domain expert to come up with these real examples and therefore is applicable for those domains that are on the ‘tangible’ side of the anchoring scale (see chapter 2).

A brief conclusion regarding the suitability of the three approaches that we have studied in this chapter of this thesis is that the EER and ORM approach do not comply to the completeness criterion for the way of modeling that was defined in chapter 2 and that contains a description of what type of ‘domain knowledge’ in essence must be incorporated in a requirements specification. Furthermore, there exists a large difference between the families of approaches and even between members within a given family in terms of the extent in which the application domain semantics can be expressed in the data model, and as static or dynamic constraints, static derivation rules or dynamic constraints. With respect to the completeness criterion for the way of working we can conclude that ORM is the only approach that provides some assurance that all relevant semantic relationships in the data model and some types of static constraints will be detected in the application UoD. This means that there still is an opportunity to improve the requirements determination approaches we have surveyed in this chapter in terms of the completeness aspects that were given in section 2.1.

With respect to the efficiency criterion for the way of modeling we must remark that in EER and UML in a number of cases remodeling is necessary not
because domain semantics have changed, but because the attribute modeling construct has been applied in the initial requirements specification. With respect to the efficiency in the way of working we concluded that in some species of the family of EER approaches modeling procedures do exist. However, they rather tell an analyst what to do next than to specify how he/she must do it. In ORM a CSDP (conceptual schema design procedure) is given that gives more guidance on how an analyst must apply the modeling concepts than in the EER counterparts. However, with respect to the derivation of static constraints ORM does not give a procedure that specifies how an analyst can find all instances of such a constraint type in a given UoD. In the defining literature of UML no (rudimentary) procedure is provided that tells an analyst how to detect instances of constraints (in a dialogue with a domain expert). With respect to the efficiency in the way of controlling we must conclude that none of the approaches (EER, ORM and UML) provides quality assurance steps and the EER and UML approaches do not provide an activity structure that gives handles for optimizing performance, cost and time.

With respect to the formality criterion for the way of modeling we can conclude that in many (E)ER approaches and in the UML, it is not possible to apply a consistent definition for minimum cardinalities or multiplicities across all types of semantic relationships. In UML it is not clear how the modelling concepts that are used in the 9 different diagram types are related on the level of an application requirements specification. In ORM an inconsistency is found with respect to naming conventions. Furthermore, we remark that the optionality or non-existence of some modeling constructs in all three approaches that we’ve studied might lead to imprecise and inconsistent requirements specifications. The non-required naming conventions for model elements in EER, UML and ORM can lead to traceability problems.

With respect to the formality criterion for the way of working we can conclude that for EER and UML the formality of the procedure is not relevant because there hardly exists any procedure. With respect to the CSDP in ORM we remark that those segments of the CSDP that can be considered prescriptive documents are at most semi-formal. In EER, UML and ORM no procedure exists that allows an analyst to question the assumptions on which the utterance of the domain semantics is based.

Table 3.4 Comparison EER, ORM and UML approaches on completeness, domain richness, efficiency and formality criteria for the way of modeling, way of working and way of controlling

<table>
<thead>
<tr>
<th></th>
<th>EER</th>
<th>ORM</th>
<th>UML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completeness</td>
<td>W o M</td>
<td>W o W</td>
<td>W o C</td>
</tr>
<tr>
<td>Domain Richness</td>
<td>W o M</td>
<td>W o W</td>
<td>W o C</td>
</tr>
<tr>
<td>Efficiency</td>
<td>-</td>
<td>-</td>
<td>n. a.</td>
</tr>
<tr>
<td>Formality</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
The position of these approaches is basically that the domain requirements that are uttered by the user are encoded in the model 1-on-1. ORM claims in steps 3, 4 and 7 to perform checks on sample populations; however, it does not give guidelines on how to perform these checks in a dialogue with the responsible domain user.

With respect to the formality criterion for the way of controlling we must conclude that there exist no formal quality assurance algorithms in EER, UML and ORM. Finally we can conclude that ORM is the only approach that has facilities for formally planning a requirements determination project in terms of the stages in the conceptual schema design procedure (CSDP).

In table 3.4 we have summarized the deficiencies from the three approaches studied. A ‘+’ denotes that an approach does fully comply with this criterion for the given aspect of the RDM. A ‘0’ means that an approach complies to some extent. A ‘-’ means that an approach does not comply at all.

After studying the existing literature on RDM’s we can conclude that no single approach fulfills the criteria that were given in chapter 2. To put it even stronger: even a compilation of approaches in which the best features of a number of approaches will be combined, will not comply with the four quality criteria from chapter 2. In chapter 4 we will use the flaws, inconsistencies and omissions that we have diagnosed in the state-of-the-art in RDM’s when we operationalize the quality criteria from chapter 2 into an operationalized design specification for a to-be designed RDM.

3.9 REFERENCES


Bollen, P. (2002c): De totale samenhang tussen de diagramsoorten in UML. Meteor research memo RM 02/048. Faculty of Economics and Business Administration. University of Maastricht. (in Dutch)


Liang, Y. (2003): From uses cases to classes: a way of building object model with UML. Information and Software Technology 45: 83-93


80


