Extending CDIF to support Business Rules targeting SQL3

Kristian Palmquist

Submitted by Kristian Palmquist to the University of Skövde as a dissertation towards the degree of M.Sc. by examination and dissertation in the Department of Computer Science.

October 1997

I hereby certify that all material in this dissertation which is not my own work has been identified and that no material is included for which a degree has previously been conferred on me.

Signed: _______________________________________

Kristian Palmquist
Extending CDIF to support Business Rules

October 1997

Key words: Business Rules, CASE, CDIF, SQL3, Triggers

Abstract

Business rules have gained attention in recent years and are now considered to be important organizational elements. Several sources in the literature argue that there are major achievements to be made with an explicit business rule focus in software engineering, e.g. promoting communication between analysts and users and accounting for changeability and maintenance aspects. However, to fully take advantage of an explicit rule focus in software engineering requires the ability to create business rule models. The problem is that business rule models of realistic size quickly become extensive and complex, hence there is a need for CASE tool support.

We choose a modeling technique from the literature which is suited to express business rules. Based on this modeling technique we propose an extension to the Case Data Interchange Format standard (CDIF), thereby allowing the standard to express and support the transfer of business rule models. In addition, we define a mapping procedure which maps business rules from the conceptual modeling level via CDIF (using the proposed extensions) to SQL3 triggers. The main idea is that the mapping algorithms could be used by a CDIF conformant CASE tool which allows traditional database design, together with extended modeling constructs for expressing business rules.
# Table of contents

## 1 Introduction

1.1 Aim

1.2 Objectives

1.2.1 Choosing a BR model

1.2.2 Extending CDIF

1.2.3 Mapping to SQL3

1.3 A scenario

## 2 Background and foundation

2.1 Introduction

2.2 What is a business rule?

2.2.1 Definition

2.2.2 Classification of business rules

2.3 Are business rules important?

2.3.1 A business rule focus in Software Engineering

2.3.2 Advantages of an explicit rule focus

2.3.3 Problems

2.3.4 Rules in information systems

2.3.5 Business rules and CASE support

2.4 CASE tools and repositories

2.4.1 Introduction and definitions

2.4.2 CASE tool/repository architecture

2.4.3 Repository functionality

2.4.4 The Meta-model

2.4.5 CASE tool/repository standards

2.5 CDIF

2.5.1 Introduction

2.5.2 Subject areas

2.5.3 Detailed description of the subject areas

2.5.4 The architecture

2.5.5 The transfer format

2.6 The database language SQL3

2.6.1 Active functionality

2.6.2 Trigger syntax

2.6.3 Trigger semantics

## 3 Choice of model

3.1 Requirements of a model

3.2 Choice of model

3.2.1 Candidate models

3.2.2 Selection

3.3 The Entity-Relationship Model (ER)

3.3.1 Semantic constructs

3.3.2 Constraints

3.3.3 The ER diagram

3.4 The Extended Entity-Relationship Model (EER)

3.4.1 Which EER variant to use?
3.4.2 Extended semantic constructs ................................................................. 49
3.4.3 Additional constraints .............................................................................. 51
3.4.4 The EER diagram ....................................................................................... 52

3.5 The Entity-Relationship Rules Model (ER2) ............................................ 53
3.5.1 Constructs for expressing rules and events .............................................. 53
3.5.2 Events ........................................................................................................... 54
3.5.3 Rules .............................................................................................................. 55
3.5.4 Actions ........................................................................................................... 56
3.5.5 The ER2 diagram ......................................................................................... 58

4 Extending CDIF .......................................................................................... 60
4.1 The CDIF data modeling subject area ......................................................... 60
4.1.1 A small example .......................................................................................... 60
4.2 Mapping EER to CDIF ................................................................................. 64
4.2.1 Semantic constructs and constraints ......................................................... 64
4.2.2 An example mapping .................................................................................. 65
4.2.3 Analysis of semantic constructs with no explicit support ....................... 71
4.2.4 Discussion .................................................................................................... 72
4.2.5 Proposed additions ..................................................................................... 72
4.3 Rules for extending CDIF ............................................................................. 73
4.4 The semantic extensions ............................................................................. 74
4.4.1 Events .......................................................................................................... 74
4.4.2 Rules ............................................................................................................. 75
4.4.3 Actions ......................................................................................................... 75
4.5 The extended Meta-model ........................................................................... 77
4.6 The detailed definition of the Meta-model .................................................. 78
4.6.1 Meta-Entities ............................................................................................... 78
4.6.2 Meta-Relationships ................................................................................... 83
4.7 Evaluation of the design ............................................................................. 92
4.8 Mapping ER2 to CDIF ............................................................................... 93
4.8.1 The data model ........................................................................................... 93
4.8.2 Events .......................................................................................................... 93
4.8.3 Rules ............................................................................................................. 94
4.8.4 Action .......................................................................................................... 95
4.8.5 Discussion ................................................................................................... 95

5 Mapping to SQL3 ....................................................................................... 96
5.1 Introduction ................................................................................................... 96
5.2 The mapping description ............................................................................ 96
5.2.1 Discussion .................................................................................................. 97

6 Discussion and conclusions ................................................................. 100
6.1 Introduction .................................................................................................. 100
6.2 Choice of model .......................................................................................... 100
6.3 Extending CDIF ......................................................................................... 101
6.3.1 Mapping EER to CDIF .............................................................................. 101
6.3.2 Extending the Meta-model ...................................................................... 102
6.3.3 Mapping ER2 to CDIF .............................................................................. 103
6.4 Mapping to SQL3 ..................................................................................... 104
1 Introduction

This work is built upon the assumption that business rules (BR) are an important aspect of an enterprise and that they should be taken into account during software development. The concept of business rules has gained attention in recent years and these rules are now considered to be important organizational elements [App84], [Ass88] and [Mor93]. An explicit business rule focus in software engineering is one way to improve productivity, quality and changeability as identified by Ross [Ros94] and Usoft [Uso97a].

It is the strong belief of this author that there are several benefits to be gained from a rule based approach to software development. This is the case regardless of whether we view business rules as a new paradigm or as a complement to traditional views of information systems, e.g. function and data oriented views. This work deals with the behavioural dimension of database driven application development in general and more specifically with models used to express business rules, i.e. we do not elaborate on data modeling even though the interrelationship between business rule models and data models is highlighted.

In spite of the fact that business rules have received attention, there is no consensus within the scientific community about the definition of the concept of a business rule. Some people regard them as production rules (originally found in AI systems from the seventies) represented in the form of IF condition THEN action. Others regard business rules as the constraints that it is possible to define on data structures [Hal94]. These structures are commonly expressed in data models together with constraints, e.g. cardinality and participation constraints over relationships and attribute-domain constraints. A third category of people regard business rules as high-level statements of how business is being done. This third category represents the view taken in this work.

In spite of the inconsistent use of the term, business rules have had an impact on the database community. Some examples are found in the recent IDEA project which aims at producing a development environment for advanced database applications [Cer97]. One can also cite the Business Rule Summit 1996 which was a conference totally devoted to Business rules [Brs96] and USoft Developer, a business rule based development tool recently released from Usoft [Uso97a]. This section provides a reference for the view of business
rules adopted for this work, and motivates the approach taken to *consider the interchange of business rule models between CASE tools*.

The view taken of business rules in this work is based upon the work of Ronald G. Ross in his in-depth book on business rules: The Business Rule Book: Classifying, Defining, and Modelling Rules [Ros94]. Others have followed Ross’s view of business rules and some examples of this are provided in the GUIDE project [Gui96] and in Database Programming & Design (DBPD) [Halb96]. Section 2.1 presents a detailed definition, classification and set of examples of business rules from the Ross perspective.

An important problem when focusing on business rules, is the lack of a consistent and systematic treatment of business rules in the Software Development Life Cycle (SDLC). To fully take advantage of an explicit rule focus in software engineering requires the ability to create rule models. We believe that business rules can become a major building block in software engineering and gain acceptance if these rule models can be handled with CASE support. CASE support for business rules is motivated by the fact that business rule models quickly tend to become complex and extensive (further elaborated on in section 2.2.5). A fully fledged CASE environment including business rule models would give the analyst/designer a powerful tool.

The idea is that the rule models would complement traditional models supporting the database design process, e.g. data models and dataflow diagrams. The tool could also provide associations between models, e.g. it is common that rules affect data model objects. The CASE tool would allow sharing of rule models between analysts/designers and it would be able to store different types of models supporting different stages in the SDLC, e.g. conceptual and logical models. Furthermore, the tool could perform automatic mappings between different rule models and generate code, e.g. trigger specifications.

Today, it is very common for companies to have multiple CASE tool environments. For the rule paradigm to be really effective, it must be possible to exchange rule models between various CASE tools. There is an increased need for standards for the exchange of information (models) between CASE tools (further elaborated on in section 2.3.5). The task of model exchange between various CASE tools is addressed by the Case Data Interchange Format standard (CDIF) [EIA106] [CDIF97a]. CDIF is not a specification of a
CASE tool or a repository; it specifies an export/import interface for CASE tools or repositories (the specific choice of the CDIF standard is also motivated in section 2.3.5). This work addresses business rule model interchange between CASE tools by extending the CDIF standard. The extensibility mechanism of CDIF is a powerful means of exchanging information that is not defined in the standardized CDIF integrated Meta-model, i.e. the Meta-model of CDIF is extended using the extensibility mechanism and can hence allow the exchange of richer information (new or extended models).

Analysts and designers need formal guidance to capture business rules (methods) and express them in various modeling techniques (models). This work only considers models with the ability to express business rules. Hence the problem of method support is outside the scope of this dissertation even though, method issues are considered to be equally important by the author. These rule models should be used in the analysis steps of the SDLC. This work defines the requirements for a conceptual rule modeling technique and chooses a model from the literature, which is suited to express business rules. The specific model is used to propose an extension to the CDIF standard to incorporate business rule models. The proposed extension is specified using the standardized extensibility mechanism. Currently, there is no support for this type of information in CDIF. The standard body has concentrated on modeling techniques which aim at database design, i.e. dataflow diagrams and data models [Elm94], however new subject areas are currently in working group reviews (e.g. business process modeling and OOAD).

In addition, this work addresses the problem of a consistent treatment of BRs in the SDLC by defining a mapping procedure, which maps business rules from the conceptual modeling level via the interface standard CDIF (using the proposed extension) to SQL3 [ISO96]. The main idea is that the mapping algorithms should be used by a CDIF conformant CASE tool which allows traditional database design, together with extended modeling constructs for business rules. It would support the automatic generation of DBMS language constructs. Furthermore, this work will address the question of the utility of the extensibility mechanism found in CDIF. It is also concluded in the literature that there is a need for conceptual models expressing business rules (section 2.2.3). Conceptual models are of great importance if one wants to communicate in business rule terms with users. This seems to be a reasonable approach, since communicating with business people in
business related terms could be an alternative to communicating in terms of data, flows and functions. The process of eliciting requirements for an information system with a BR approach seems to be interesting, however it is not the purpose of this work to consider these early stages in the SDLC. Note that this work does not elaborate on new or revised modeling techniques for expressing BR.

1.1 Aim

The aim of this work is to propose an extension to the Case Data Interchange Format standard, so that it can express and support the transfer of business rule models. In addition, we define a mapping procedure which maps business rules from the conceptual modeling level via the repository standard CDIF (using the proposed extensions) to SQL3. Figure 1 introduces this idea graphically. The mapping procedure only treats a specific type of business rule.

Figure 1: Aim and objectives
1.2 Objectives

1.2.1 Choosing a BR model

The first objective is to define the requirements for a model for expressing business rules. We search the literature for suitable models that fulfill these requirements. A choice of one specific modeling technique is made for representing business rules. The motivation for the choice of that specific model is further elaborated on. Every atomic semantic construct and constraint of that model is listed. We also show why the current standardized modeling techniques in CDIF are insufficient for expressing business rules.

1.2.2 Extending CDIF

The second objective is to propose an extension to the CDIF integrated Meta-model that can incorporate the semantic constructs of the chosen rule model. The extensions have to follow the rules stated in the standardized subject area of extensibility. The design process is carried out by systematically creating a corresponding semantic construct in the CDIF integrated Meta-model (by adding a Meta-entity, Meta-attribute or Meta-relationship), for every atomic semantic construct in the chosen model. This work claims systematicity, not completeness in the mapping procedure. This means that we cannot necessarily account for all combinations of atomic constructs. It is the opinion of this author that this restriction does not severely limit the usage of the mapping procedure (presented in the last part of the second objective below), since most models used in every day practice do not possess these characteristics. All design decisions should result in extended semantic constructs in CDIF that allow for a one-to-one (1:1) mapping of a business rule model to CDIF. This requirement results in the ability to use the mapping procedure backwards.

The last part of the second objective is to define a mapping procedure that uses the extended Meta-model. This work is strictly analytical, i.e. it does not propose and implement a database system. The alternative would be to implement the CDIF integrated Meta-model in a relational database system. However, this approach would not yield any additional information. The cardinality and participation constraints on the integrated Meta-model can be tested without performing a mapping to a relational model. The CDIF Integrated Meta-model can also be used to design a repository. This is not the purpose of this
work and it should not be confused by the fact that the integrated Meta-model is extended (the Meta-model produces the transfer). This implies that any comments regarding how well suited the CDIF integrated Meta-model is to provide the basis for a repository is not considered to be of interest, i.e. we will stay with the original goal of CDIF as a transfer format by focusing on issues relating to the preservation of semantics from the chosen model aiming for minimal extensibility.

1.2.3 Mapping to SQL3

The third objective is to specify a procedure for mapping the CDIF representations, i.e. the extended Meta-model expressing the business rules, into triggers in the forthcoming SQL3 standard. The mapping procedure only regards triggers without calls to stored procedures. The reverse mapping between the triggers via the CDIF representation to the conceptual business rule model is guaranteed by the fact that the design of the Meta-model only takes into account constructs that allow for a one to one (1:1) mapping.

1.3 A scenario

How does all this fit together? Why the interest in an interchange standard? How is CDIF going to be used in practice? What are the benefits of using the standard? Why can we not use vendor specific bridges between all tools for our business rule models?

Let us consider a scenario that explores these issues. Assume that we have a multiple CASE tool environment. The tools involved have the ability to create and edit business rule models. In order to exchange rule models developers would have to recreate models in a target tool. This is a very costly and time-consuming activity. It is very likely that it would introduce errors and also require experts of the source tool as well as the target tool.

One solution would be for all tool vendors to specify export/import interfaces to all other tools. This would result in a large number of interfaces (even if feasible) and it is likely that the vendors would support only the major competitors, or worse, they would only allow importers! In addition the vendors would need to track every change in the other vendors’ Meta-models in order to guarantee successful transfers at all times. The scenario where a vendor can exchange information only with a few other major vendor tools can become a difficult problem. It has been shown that it is crucial for an organization to
choose a CASE tool based on their specific requirements [Bub88]. This issue is addressed by the ISO standard for evaluation and selection of CASE tools [ISO97].

The alternative scenario is that tool vendors adhere to CDIF. The business rule Meta-models would be different in the various tools/repositories (compare with different variants of data models, i.e. different Meta-models). Then, in order to transfer business rule models between these tools the vendors would have to adhere to the CDIF integrated Meta-model that expresses business rule models. This would imply that the vendors have to specify one export/import interface that conforms to CDIF. This guarantees that the various tools could exchange business rule models in a uniform and consistent way. This scenario is presented in Figure 2\(^1\) [CDIF97b]. The benefits of this scenario are that the developers could choose tools with different capabilities, change tools and versions and always be sure of an information consistency between all tools used. If CASE tool vendors adhere to CDIF this guarantees that it is possible to exchange information between tools.

---

1. The picture is adapted from a presentation by Johannes Ernst who is a technical officer at CDIF. It has been modified to reflect rule models.
What is the trend in CASE tool usage? Is the exchange of models between various tools a major problem? Steven Kelly, one of the developers of Meta-edit [MET97], considers the question of the trend in CASE tool usage:

“The basic answer is that CASE is growing and will continue to grow [Kel97]”

He supports this statement, for instance, with the findings of Kusters and Wijers [Kus93] that 89% of ISD managers see the future importance of CASE in their organization increasing. Since the CASE tool market is still growing this implies that the exchange problem must be addressed. In addition, Stobart et al. [Sto93] summarize their world-wide findings of problems related to CASE tool usage. One of their main findings is directly related to this work:

“The integration between tools for different software development phases is poor.”

It is the opinion of this author that the CDIF standard addresses this problem in an interesting way and this has been a driving factor for this work. Readers interested in CDIF can use this work as a compact introduction to the standard (section 2.4) together with examples (section 4.1 and 4.2.2).


## 2 Background and foundation

### 2.1 Introduction

This chapter contains an extensive overview of issues relating to this work. It provides a basis for the project and ties the issues together, forming a background and foundation for a business rule extension of CDIF and a description of mapping business rules from the conceptual modeling level to triggers within SQL3. Section 2.1 defines a business rule with respect to this work and presents a classification scheme for different rule types. Furthermore, it provides a number of examples of business rules for every type found in the classification scheme. Section 2.2 elaborates on the need for and achievements of an explicit business rule focus in software engineering. It also identifies major problems on the journey to a rule-based paradigm in software engineering and emphasizes that CASE tool support is crucial. This section also emphasizes the need for a consistent treatment of business rules throughout the software engineering lifecycle and the need for conceptual models of business rules. Section 2.3 introduces CASE tools and repository architectures. This is essential since there is confusion about the definition and role of these products. The aim of section 2.3 is also to clearly state the view of these products taken in this work. Section 2.4 presents the Case Data Interchange Format standard (CDIF). The presentation includes the historical development of the standard, the ultimate goals and the architecture of the standard. Finally, section 2.5 elaborates on active functionality in the forthcoming database language SQL3, i.e. triggers. It presents the arguments for a centralized implementation of rules and elaborates on the structure used to represent rules. It also includes some comments on assertions, including an argument of why they where not considered in this work.
2.2 What is a business rule?

2.2.1 Definition

The term business rule is used in a broad sense in this work, but it has been used differently throughout the literature, often being restricted to semantic integrity constraints [App88]. The definition of a business rule with respect to this work is:

**Definition 1:** Business rules are statements how the business is done, i.e. guidelines and restrictions with respect to state and process in an organization [Ass88].

2.2.2 Classification of business rules

Probably the most extensive classification made of business rules is that of Ronald G. Ross [Ros94]. This work is built on his view of business rules and this section presents his classification scheme. Barbara Von Halle [Hal96a] has noted that Ross’s view of business rules coincides with the Zachman framework for information system architecture [Zac92], in that a business rule means the natural-language expression used in the business perspective [Zac92, P600 (row 2)]. A rule means an implementable and technology independent piece of logic [ZAC92, P600 (row 3)] and they can be further transformed into technology-dependent specifications [Zac92, P600 (row 4)]. The terms business rules and rules will be used synonymously in this work, and the context will determine the type of rule discussed.

Ross’s scheme is based on a few basic rule types that can be combined into complex business rules. There are nine basic types of atomic business rules presented below. In addition there are a number of subtypes and operators on every basic rule type. However, it is not my intention to elaborate on every subtype and operator of every atomic rule type, since this material is too extensive. Those readers further interested in business rules are highly recommended to study Ross’s book which also provides over 500 example rules. These examples also include a diagrammatic representation in the Ross Methodology, which is an extended ER model with additional constructs for expressing rules. The first rule of **instance verifiers** below refers to the attribute type “date” of the entity type “order” and the first rule of **type verifiers** refers to an IS_A relationship between “customer” and its subtypes “government”, “corporate” and “individual”. All the examples in the nine cate-
Background and foundation

gories are collected from Ross’s book and they all have a graphical representation in the book.

1. **Instance Verifiers**
   
   This type states the possession of instances, i.e. an instance of a constrained object may exist only if the indicated test for possession is satisfied.

   **Example rules**
   
   “Every order must have a date placed at all times”
   “An order may be placed at most by only one customer”
   “To hold a credit account a customer must be “good””

2. **Type Verifiers**
   
   This type states the existence of a given type, i.e. the rule evaluates the existence of a given type. This is distinguished from the instance verifiers which always count actual instances of a constrained object.

   **Example rules**
   
   “A customer may be either a government, corporate or individual, but never more than one of these types.”
   “An active project must have a budget, and vice versa”

3. **Sequence verifiers**
   
   This type states the changes of instances of constrained objects and usually involves a series of updates. This type of business rule always elaborates on a logical sequence, referred to as an explicit logical sequence.

   **Example rules**
   
   “An order must be indicated as received before any other statuses can accumulate”
   “If the most recent status upgrade for a contracted project is not step-wise, then...”

4. **Position selectors**
   
   This type states an innate sequence (an implicit sequence) of a database that can be based on a value or the age of an instance. If a business rule refer to an arbitrary sequence then the sequence verifier should be used instead.
Background and foundation

Example rules
“If this section of the report is the first section then...”
“If this is the fifth section in the report, then...”
“If this is the order with highest total order amount of all, then...”

5. Functional evaluators
This type of rule test values of constraining object(s) as a function of the instances of the object.
Example rules
“All order must be identified uniquely”
“The same effective date never may be used more than once for all orders”
“Rental agreement periods must vary by at least 3 days from any other”

6. Comparative evaluators
This type of rule performs a comparison between similar values of objects.
Commonly, the values exist of attribute types.
Example rules
“Approved orders require that the total order amount must be less than or equal to the customer’s credit limit”
“If the balance of the account falls below USD 1000, then...”
“A customer may be supported only by an agent in the same city”

7. Calculators
This type of rule tests the result of a common computation of values on constrained objects. Commonly, these values are attribute types.
Example rules
“The amount owed for an order must be the sum of the prices of all products on the order”
“If the amount of a withdrawal is above the average for the account, then...”

8. Update controllers
This type of rule defines constraints on database updates or what result should be produced. This is applied to all instances of a constrained object (except when enumerating the instances), as opposed to instance verifiers that perform
the same operations but with the default “lower limit of one”

**Example rules**

“Once an account is created, its type never must change”

“An order must not be modified in any way, or deleted, once closed.

“If the status of an employee has changed, then...”

9. **Timing controllers**

This type of rule tests intervals of time on constrained objects. Timers may also be used to switch off and on other rules.

**Example rules**

“An order must not be deleted for at least one minute after it has been created”

“An order must be deleted automatically 30 days after completion”

“An employee must not assist project for more than one year total”

A complementary classification scheme proposed by Barbara von Halle states four types of business rules, which are presented below [Hal96b]. A business rule can fall into one of the four categories:

- **Definitions of business rules**
- **Facts relating terms to each other**
- **Constraints**
- **Derivations**

Responding to a direct question made by this author of the exact relationship between the two classifications, she states:

“We (Ross and von Halle) both use the term Business Rule (as does the GUIDE business rule project) to refer to Terms (nouns with definitions), Facts (connections among nouns), and Rules (the testing of conditions). My constraints and derivations fit under the heading of Rules. Ross periodic table of ruletypes also fits under the heading of Rules. My classification is at a higher level and he gets down to the atomic pieces based on the kind of “work” or “computation” that a rule performs.”

[private email communication, 97-08-09].
Definitions of terms in a specific language state a large amount of information concerning how we perceive things. This is the most basic aspect of a business rule and it is often described as entities in Entity-Relationship models. An example of a definition is an invoice, which a person probably associates with a number of characteristics, e.g. “it contains information about a customer” and “it contains information concerning delivered goods and prices”. Facts relating terms to each other express business rules that defines the structure of the information in a company. The facts are often described as attributes, relationships or generalization structures in Entity-Relationship models. An example of a fact is that “a customer can hire a car”.

It is common that a company has defined constraints on the possible operations of stored information. This class of business rules are actually the inverse of the facts, since they state what can not be done while the facts state what is possible. The constraints often refer to definitions and facts. Some of the constraints are possible to express in ER models, e.g. integrity constraints and domain constraints while some constraints have to be expressed in other formalisms or extended ER approaches. One example of such constraints is that “no employee is allowed to have a higher salary than the executive”. Derivations are derived from other facts and specifies new information. One example of a derivation rule is “the weekly charge is the daily rate times seven minus the discount”. Note that these types of rules assume an ordering when we obtain the information, i.e. we must know “the daily rate” before we get to know “the weekly charge”. This ordering has to be defined by the analyst.

### 2.3 Are business rules important?

All enterprises are guided and controlled by a huge amount of business rules and they are the very foundation of structure and actions. The GUIDE project [Gui96] states:

> “It is important to note that an enterprise’s business rule applies, regardless of the form used to express it. Business rules have been in place and companies have been responding to them long before anyone ever dreamed of formalizing them. Business rules are an underlying reality in an organization, independent of an analyst’s attempt to structure and describe them.”
Von Halle [Hal97] also emphasizes the fundamental importance of business rules:

“No matter how you capture it, the logic behind business requirements actually represents a set of business rules.”  

Now, assume that a company suddenly would lose all its business rules. It is plausible that the company then would have to struggle to stay in business, i.e. without business rules a company would probably not stay viable for long. What is the nature of these business rules? Some are very “high-level” policies or specifications of routines used in the daily management of the enterprise. It is common that rules are specified in various documents, e.g. working descriptions. It is important to note that the rules incorporate both structural and dynamic aspects of how business is being done. Examples of the origin of rules could be company goals or processes. Some rules may be informal and general, forcing the employees to interpret them and some may be reasonably formal.

Probably, a large subset of all the rules have an information system component. It is these types of rules that will be treated here. These rules typically places constraints on database states, derives new information or executes actions when certain events occur. The usual treatment of business rules throughout the software development process is that they are implicitly expressed in various models and documents, and not explicitly attended to. Just recently there has been an increased interest in the explicit treatment of business rules in systems development [Kno94].

2.3.1 A Business rule focus in Software Engineering

There are many different ways for a systems analyst/designer to describe an enterprise, each way representing a specific view, which helps the developer to model the interpretations of the UoD. There are well established methodologies for this work, e.g. function oriented, data oriented and event oriented methods. With these methods it is possible to describe how information flows, functions are performed and data structured within an enterprise. It is easy to see the need for modeling of different views of the enterprise for specifying requirements and designing Information Systems. All these views are “pieces of the puzzle” which promotes the development of good quality systems.
Loucopoulos and Kopankanas [Kop89] states: “The knowledge found in an application domain can be varied and with many facets and it is a truism that there does not yet exist an ‘ideal’ representation formalism that possesses logical adequacy, heuristic power and notational convenience that is needed for encoding satisfactorily all types of domain knowledge.” (P. 1).

However, is it possible that a dimension is missing? Several authors believe that there is a missing view and their view is based on the concept of business rules [App84] [Mor93]. There are even authors who call for a “paradigm shift” in favour of a business rule focus in information systems development [Ass88]. A business rule view complements traditional views on information systems, as stated by Barbara von Halle [Hal96c]:

“Business rules bridge the gap between process and data; between behavior and information” (P. 12).

A business rule focus would imply starting off with an analysis of user requirements, goals and business policies and eliciting the business rules from these. First after this procedure, an analysis whether the rules should be stated in data models or as separate business rules in a rule model would be performed. See figure 3 for a presentation of these ideas created

![Figure 3: A synergy between business rules and data models](image-url)
by Barbara Von Halle [Hal96b, p.16], where she proposes a synergy between business rules, data models and object models. In her opinion it is a good idea to start off with an analysis of present business rules, to classify those and finally create data models and rule models. This allow definitions and facts to be expressed in data models, expressed as entities, attributes and relationships, together with some constraints, e.g. cardinality and participation constraints on relationships between entities.

However, data models lack the ability to express dynamic behaviour, and therefore the remaining constraints and all of the derivations would be expressed in rule models in other formalisms. This is an alternative to the traditional ways of working, e.g. when focusing only on data modeling we could try to find the relevant concepts with respect to the future information system, their characteristics and relationships between concepts, which through refinement eventually would specify the design of a database. This would capture the business rules that are possible to express in traditional data models. However many business rules are not possible to specify in a data model. They would probably be implicitly expressed in various models or not modelled at all until it is time to convert the rules into code.

Probably a number of the business rules would be implicitly expressed in various process models and thus inconsistently implemented (scattered) into procedural code. This makes it very hard to know what business rules that apply to the system and where they are implemented. It would be very hard to accommodate for quick changes. This work is focused upon the constraints and derivations that are impossible to express within traditional data models. In contrast to these ideas recent advances in the area of active databases have put a “rule focus” on the design of databases, see e.g. the conference on rules in database systems [RIDS97a].

2.3.2 Advantages of an explicit rule focus

Sources in the literature argues that several achievements could be made if an explicit rule focus is adopted in software engineering, i.e. incorporated into methods and modeling techniques. There are proposals that business rules serve as a good base for communication between system designers and business people (users), as stated by Loucopoulos [Lou95]:
Background and foundation

“The major efforts in addressing business policy modeling involve the use of a rule-based paradigm which provides a natural mapping from enterprise to information systems concepts.” (p. 100).

These ideas are also consistent with those in [Hal96b] which state that business rules express definitions, facts, constraints and derivations in the same language business people would use to describe them. The idea is that it is easier to communicate with business people in terms of business rules, rather then in information systems terms, e.g. data, data flows and processes. It has also been showed that the rule paradigm is a good means for expressing business knowledge [Ass88].

In addition to promoting communication, probably the most important achievement of an explicit business rule focus concerns changeability and maintenance. Ross argues that there has been too much attention on speed and elegance in implementation of systems, and he argues that real business challenge is about changeability [Ros94]. And indeed, maintenance is a timeconsuming and expensive problem identified in the literature. A business rule focus in software engineering may very well be the answer to this problem. USOFT argues in a white paper [Uso97b]:

“People usually explain why maintenance is so difficult by explaining the complexity of technology, that’s only right maybe 90% of the time. The right answer is that they never bothered to unify the business rules in the first place. They don’t know what business rule are implemented, so they don’t know how to find and change them.”

If business rules are neglected and only stored implicitly in various models, they are practically impossible to find and altered in the application code. However, if there is support for business rules throughout the software engineering lifecycle, starting off with conceptual models of rules, then changes in the organization with an impact on the information system can easily be accounted for. USOFT states (in the same paper):

“When the programmer leaves and the Information Technology group is left to maintain the system, can they do it? Can they find the rule, assess the impact and make the change directly, and know what the implications are? That’s the sign of a quality software product that is truly addressing a business need, as opposed to a
Background and foundation

"programming need. Business can then concentrate on the business instead of the process of the business.”

An explicit treatment of rules can also be used to fully take advantage of the power of active functionality in databases, as stated by Herbst [Her95]:

“To fully use the potential of these rule-based mechanisms, a rule-based systems analysis methodology seems necessary.”

It has been noted that this active functionality is well suited to implement business rules in a wide variety of applications [Bou97]. Common methodologies for analysis and design of databases are insufficient for the new active mechanisms found in research prototypes and commercial DBMSs [Tan92] and [Sim95]. Since there is little or no support in Software Engineering methodologies and modeling techniques, the designers have to use ad-hoc solutions to incorporate the active functionality late in the design process. The problem with this procedure is that the implementation model easily can bias the semantics from the real world [Tan92], i.e. there is no conceptual modeling of rules.

Business rules usually constitute a main part of an information system and are scattered all over the system, often deep buried in procedural code. There is a need for the benefits of a centralized and non-redundant implementation of business rules. This development calls for a systematic and consistent treatment of rules throughout the software engineering lifecycle. It implies new or revised methodologies for conceptual design of active databases, since the existing methods were developed for passive databases [Tan92]. In the area of active databases the focus has been set on the more technical aspects such as event languages, transaction models and rulebase management. The conceptual modeling of rules has not gained a lot of attention and rules placed in the context of software engineering even less - something which has been discussed recently [RIDS97b]. However, to effectively develop active databases requires that these issues are addressed, e.g. as in this work.

2.3.3 Problems

Is it straight forward to adopt an explicit business rule focus in software engineering? The answer to this question is; no. There are a variety of important issues that needs to be
resolved in order to efficiently use the business rule paradigm. The first and most basic issue regards the definition and classification of business rules. There is no consensus within the database community on what a business rule really is. For the rule-paradigm to gain acceptance it is necessary to achieve this consensus.

The second issue regards the lack of support for business rules in software engineering methods and modeling techniques. Analysts and designers need formal guidance to capture the rules (methods) and express them in various modeling techniques (models). These models need to be refined throughout the SDLC. An example of a business rule analysis scenario is provided by von Halle [Hal96d]. It is founded on the Zachman framework [Zac92], and describes the steps one at a time. Von Halle also discusses the problem of business rule support in different software engineering methods (waterfall-based, prototyping-based and OO-based methods) [Hal97].

This work only consider models with the ability to express business rules and hence the problem of method support is outside the scope of this dissertation. It has been argued in the literature that there is a need for conceptual models of business rules [Kop89], [Lou90], [Kno93] and [Ros94]. Conceptual models offer a number of advantages and are frequently used in software engineering. They are a special type of models which generally are characterized by their high-level nature, which means that they are closer to how users perceive the world. Furthermore, a conceptual model is free from implementation details, thus the models are fairly easy to understand and serves as a base for communication. This work addresses the need for conceptual models expressing business rules by identifying such approaches in the literature. Furthermore, we compare the candidate models for expressing business rules. The result is the choice of one specific model (in addition based on other criteria than merely being conceptual - see section 3.1) which serves as the base for an extension of CDIF to incorporate business rules. The last issue regards the mindset of those interested in business rules. The developers need to recognize that business rules are important for a specific project, e.g. the company wants to trace the rules to implementation and deployment or it should be possible to make quick changes among the rules. We think that these problem will be addressed when enterprises acknowledge that business rules are an important asset. This may take a while, but bear in mind the time it took before companies recognized that data was an important asset.
2.3.4 Rules in information systems

Only a subset of all rules that control the business are relevant with respect to the information system. As mentioned above, they can be recognized by the fact that they have at least one or more rule components that reside within the information system. See figure 4 for a description of possible locations of rules. The rule-set marked with nr. 1 represents rules that are not interesting to model with respect to the information system. Examples of such rules can be human behaviour rules and norms which indeed are very important for companies, but have no direct connection to the information system. The rule-set marked with nr. 2 represents rules that are not possible to formalize. Examples of such rules can be general policies and guidelines. The rule-set marked with nr. 3 represents rules that should not be modelled due to their inherent flexible nature. The rule-set marked with nr. 4 represents rules that reside within the application code. Examples of such rules can be that a customer should only be allowed to place orders if he/she has got the sufficient credit rating. The rule-set marked with nr. 4 represents rules that reside within the database. This means that the database system enforces the rules. Examples of such rules can be that no employee is allowed to have a higher salary than the executive. Note that the examples of rule-sets 4 and 5 can be interchanged, i.e. it is up to the analyst where he/she wants to specify the rules. A discussion of pros and cons of placing rules in the database compared
to the application programs will be one of the issues in the chapter dealing with the database language SQL3.

2.3.5 Business rules and CASE support

As mentioned above, we believe that the explicit modeling of business rules can become a major building block in software engineering and gain acceptance if rule models can be handled with CASE support. So, what are the specific arguments for the need of tool support for business rule models? Herbst et al. argues that rule models of realistic size quickly become extensive and complex [Her94]. Recall that we argued that an enterprise is guided by a huge amount of business rules which need to be attended to in the development of information systems. Also, for a rule paradigm to be effective the process of designing business rules must be incorporated into the traditional information systems development and database design process.

Figure 5 presents these ideas graphically. The original ideas of the figure is presented by Elmasri/Navathe [Elm94] and in addition this author have added the business rule dimension. In this figure the database design process incorporates various models in a three layered approach. Commonly the conceptual model is an ER variant, the logical model is based on an abstract relational model and the physical model is the database scheme expressed in DDL statements according to a vendor specific database system.

A common feature is that the tool can produce DDL statements for a variety of vendor specific databases. In addition it is possible to perform bi-directional mappings of the various models. This is useful to quickly accommodate for changes from the conceptual level (the data model) to the physical model (the DDL statements) or vice versa. A typical mapping from a conceptual data model to an abstract relational model would include deriving tables from entities, columns from attributes etc.
The mapping procedure could be automatic or semi-automatic, e.g. there are a variety of mapping alternatives when considering an IS_A relationship [Elm94]. This distinct layered approach of database design is now gaining acceptance in modern CASE-tools, e.g. S-Designor from Powersoft [POW97]. The process of designing business rules would be similar to the database design process, and carried out as a parallel activity. This would imply creating conceptual business rule models, logical business rule models and finally vendor specific DDL statements, e.g. triggers. In this scenario of the design process, business rules can play a crucial role in the development of information systems. The tool would provide automatic or semiautomatic mapping of rule models from the conceptual
modeling level to executable code statements. Changes could easily be accounted for by automatic support for their implications, e.g. if we change the conceptual rule model we could quickly derive the new modified DDL statements and vice versa. In this scenario rule models would complement traditional models supporting the database design process. The tool could also provide associations between models, e.g. it is common that rules affect data model objects. An example of this is the business rule “orders are only accepted if the customer has a sufficient credit rating. Obviously, this business rule involves the attribute “credit rating” of the conceptual data model entity “customer”.

2.4 CASE tools and repositories

2.4.1 Introduction and definitions

In order to explain the role and meaning of a CASE-tool/repository with respect to software development, it is a good idea to start of with a discussion of CASE tools, since they are one of the main reasons for the increased interest in repositories:

**Definition 2:** A CASE tool is an application that supports the developer of software throughout the different stages of the software engineering lifecycle [Lou95].

The aim of CASE tool usage is to increase the productivity and quality in software engineering. It also assists the developer in organizing huge amounts of information about the application development. Commonly, a CASE tool supports one or more methodologies, which traditionally have been the structured approaches, e.g. DeMarco [Dem79]. In recent years other methodologies are also supported, e.g. Object Oriented methods. When a method is supported this means that the CASE tool enforces method-specific rules. There are also CASE shells with which a user can define his own method and enforce its specific method rules, e.g. metaedit [Met97a]. There are now a wide variety of different tools with different focus. However, common for most of them is the possibility to create models, edit text and generate code. There are different types of classifications of CASE tools in the literature. One classification is based on the distinction between an early and a late stage of systems development. This is referred to as upperCASE and lowerCASE respectively. UpperCASE supports the analysis of domains resulting in a requirements specification and uses techniques like dataflow diagrams and structure charts. LowerCASE
supports the design of the information system and generation of code. It usually starts off with the conceptual model of the system which has a formal foundation as opposed to the formalisms used in requirements modeling [Lou95]. LowerCASE tools typically support the automatic generation of SQL statements from conceptual models. There is also the notion of integrated CASE tools (ICASE) [Lou95] which supports the entire development process. The development of ICASE was one of the reasons behind the increased interest in repositories. There was a need for a central storage facility, which would provide sharing of engineered objects (models, code, documentation etc.) between the earlier and later phases in the software engineering process. However, maybe even more important was the “CASE tool explosion” which denotes the quick growth in the number of available tools on the market. Since tools usually are specialized in different tasks, organizations often have a multiple CASE tool environment making it difficult to share objects between tools. The reason for having a multiple CASE tool environment is that no tools provide support for all the stages in the SDLC. Even if they did, different tools would be specialized in different tasks. Organizations began to realize that a lot of time was spent translating models from different tools and that it was difficult to reuse models. Also, since many CASE tools did not permit the existence of multiple model versions a vendor independent repository would provide support for this feature [Tan95]. A repository was the solution to this problem since it provides a central storage facility which can be shared among many different CASE tools.

**Definition 3:** A repository is a shared database of information about engineered artifacts [Ber94].

The term repository is used rather inconsistently among both users and vendors, so to complement the definition above consider the following description [Ber94]:

“...it’s implementing a layer of control services on top of the DBMS, called a repository manager, and integrating it with many tools. The result of this integration is a framework for metadata management, called a repository system.” (p. 1).

A repository can be viewed as a stand alone product, however it is common for CASE tools to include a repository in their architecture. Hence, the CASE tool is built on a repository which store the different engineered objects. These objects can be used by the differ-
ent activities in the tool, e.g. creating a conceptual data model or a logical data model. The repository must have an import/export facility which specifies the external representation of the internal meta-data in the repository. This allow the tool/repository to be able to import/export the engineered objects to other tools/repositories.

2.4.2 CASE tool/repository architecture

There is no specification on exactly what a repository architecture should consist of, however a least common denominator usually exists of [Tan95]:

- The Repository Metamodel: It represents the description about all the information stored in the repository. It is the Meta-model that express what kind of information the repository can store. The Meta-model should be extensible, i.e. it should be possible to extend the Meta-model to be able to store new types of engineered objects.

- An underlying DBMS: The contents of the repository is defined in the various Meta-models and implemented in a database. The rationale for using a database is the organization of huge amounts of information and DBMS functionality, e.g. the possibility to perform queries. The database can be relational or Object-oriented.

- Repository-Supplied Utilities: The repository functionality that extends that of the underlying DBMS. An example is versioning.

- Repository security: The security depends on the degree of separation from the repository and the database. An example is repository policies.

Basically, what distinguishes a repository from a database is the integrated Meta-model. It is the Meta-model that defines what information that can be stored in the repository. Since a repository need to be extensible, there is a need of four levels of abstraction of the Meta-models (see table 1 on page 35) adapted from [Tan95]. The bottom level is the operational level which defines the instance data, e.g. a specific customer “Mr. Green” and constitutes the production database. The next level is the data about the data, e.g. the entity type CUSTOMER that defines the characteristics of a CUSTOMER. For example, it can include information of the attributes of CUSTOMER, datatypes and recent updates. The third
level is referred to as the Meta-model and describes the elements of a particular design notation together with the rules of that model, i.e. the Meta-model defines the modeling language. It is called Meta-model since it is in fact a model about a model. This level represents the “tool’s perspective”. Finally, the fourth level of abstraction is the Meta-Meta-model which defines the model of the Metamodels components, e.g. objects and relationships. This level represents the “repository’s perspective” and it specifies a tool independent Meta-model.

2.4.3 Repository functionality

The basic functionality of a repository is defined by the underlying DBMS and include traditional database functionality, e.g. a data model, views, integrity control and queries. In addition, a variety of functions are specific to the repository itself. Examples of such functionality is stated in [Ber94] and includes the ability to check-out/check-in of objects. This means that a persistent lock is performed on the object in use. Other functions are version control, which aims at managing the different versions of an object throughout it’s lifecycle. Since everything in a repository can relate to everything a notification mechanism is needed which handle change control. These examples only represents a subset of needed functionality of a repository. The best way of implementing this functionality is to make the control userdefined, which makes it more flexible.

2.4.4 The Meta-model

The Meta-model is the basic building block for data integration and this work elaborates extensively on the Integrated Meta-model of the repository standard CDIF. So, what are the characteristics of a Meta-model and why has there been an increase in interest, for Meta-models in recent years? A Meta-model can be used for a variety of tasks, e.g. a conceptual scheme for repositories, a definition of a modeling language or a transfer format that aims at integrating tools. It usually consists of a small subset of constructs expressed in some modeling technique with the possibility to model meta-classes/entities, meta-relationships/associations and meta-attributes. These constructs makes it possible to formally define a modeling language of some sort. There are some important requirements on a Meta-model. First of all, it has to cover everything in the modeling language. This may seem obvious but even simple modeling languages include many constructs and combina-
tion of constructs, e.g. consider relationships in an ER model; n-ary relationships, constraints on relationships, relationships on relationships etc. Second, the quality of the model must be good, meaning that it should be capable of storing your model without loss of semantics. Third, the Meta-model need to be extensible since there will be situations when it is insufficient. Consider, for instance, special requirements in real life projects, e.g. including timing information into a dataflowdiagram. Also, the extensibility must be straightforward. This can be tested by adding your semantic constructs into the Meta-model and see if it fits naturally and does not disturb other elements. Finally, integration of Meta-models are important since the better it is integrated with other Meta-models from other areas, the more stable and “futuresafe” it will be. So, why have Meta-models become so interesting lately? Some ideas around this issue are expressed in [Met97b]:

“Metamodeling has been around for at least 10 years, but with the advent of the Internet and particularly the Intranet, data integration is something that is seriously getting attacked now”

Also, as elaborated on in the section on repositories, a lot of the new technologies like CASE shells (based on an extensible Meta-model), repositories and dataexchange between tools are all based on a Meta-model.

2.4.5 CASE tool/repository standards

The CASE tool market exploded in the number of available tools in the late eighties. Today, there are hundreds of different CASE tool products from a great number of vendors. These tools have greatly increased productivity, quality and reusability [Ban91] in software engineering. Many companies possess a multiple CASE tool environment and this has led to severe problems in the exchange of information between tools. As the need for exchange of data between CASE-tools have grown, this has put a pressure on the development of standards for repositories. In order to share data between tools there has to be an agreement on the Meta-models that represents the data. Otherwise every tool vendor would have to create export and import facilities for all the other vendors CASE tools. If the vendor community can agree on a specific repository standard or a standardized exchange format between tools/repositories, every tool vendor only need to specify one
import and export facility to share models with other CASE tools. This would obviously ease the integration of CASE tools and cut costs. The most common standards today are:

- CDIF (Case Data Interchange Format - EIA)
- IRDS (Information Resource Dictionary System - ISO)

These two standards are different; CDIF specifies the export/import of a CASE tool/repository and IRDS specifies the requirements for a repository. This project elaborates only on the repository standard CDIF. This has been a prerequisite and the motivation for the choice of this particular standard is as follows. Recently, there has been an increase of interest in CDIF and it is currently being merged into ISO’s standardization work. CDIF is a family of standards and provide support for the most common used methods in the software engineering process. The integrated Meta-model of CDIF is extensible, making the standard very flexible. Furthermore, it is vendor and method independent. CDIF is also currently being supported by some important vendors, e.g. Oracle, Powersoft and Unisys. CDIF is also working with the OMG standard group and UML [RAT97]. The repository standard IRDS does not deal with the exchange of data between CASE tools, rather it defines the requirements of a repository. The IRDS2 [IRDS97] standard is currently undergoing work and the original IRDS is seen as out of date [Tan95].

2.5 CDIF

2.5.1 Introduction

Originally, CDIF was a joint venture between major CASE tool vendors and user organizations. It was adopted by the Electronic Industries Association (EIA) in October 1987 and recently also by the International Standards Organization. The main idea of CDIF is to define Meta-models according to an entity relationship modeling paradigm. Different modeling techniques, such as data modeling, data flow modeling etc. can be transferred in a uniform way between CASE tools if vendors adhere to the standard. CDIF is not a specification of a CASE tool or a repository; it specifies an export/import interface of a CASE tool/repository. The interface exists of an external representation of a CASE tool’s internal data. Figure 6 presents the internal view of a CASE tool that adheres to CDIF. CDIF is not a single standard, rather it is a family of standards. The CDIF family of standards
include an integrated Meta-model and a transfer format definition. It can be divided into three groups where the first group defines the architecture, the second group defines a transfer format and the last group defines the integrated Meta-model (see below). The integrated Meta-model defines what type of modeling techniques that can be used. Furthermore, the integrated Meta-model is divided into subject areas (SA). These subject areas define data definitions, common constructs in various models and a variety of modeling techniques. Every subject area is defined in a separate document. This permits the CDIF standard to evolve over time to include more modeling techniques. It is important to note that the subject areas represents views of the Meta-model, not partitions. This means that a concept is only defined once in the Meta-model, even if it is used in several subject areas.

---

2. Picture 6 and 7 are adapted from a presentation by Johannes Ernst who is a technical officer at CDIF [CDIF97b].
2.5.2 Subject areas

This section presents the subject areas within CDIF. Note that figure 7 specifies the work of CDIF, e.g. “work focus” in the figure does not specify the focus of this work.

![Figure 7: The CDIF subject areas](image)

The CDIF architecture is given below together with references to specific subject areas in the standard (* denotes that the subject area is not a released standard yet). References to the standard documents is only provided if it is used in this dissertation:

**Defining the CDIF architecture:**

"CDIF CASE Data Interchange Format - Overview" [EIA106]

"CDIF / Framework for Modeling and Extensibility" [EIA107]

**Defining the transfer format:**

"CDIF Transfer Format / General Rules for Syntaxes and Encodings" [EIA108]

"CDIF Transfer Format / Transfer Format Syntax SYNTAX.1" [EIA109]

"CDIF Transfer Format / Transfer Format Encoding ENCODING.1" [EIA110]


"CDIF Transfer Format / OMG/IDL Bindings MIDDLEWARE.1" *

**Defining the CDIF Integrated Meta-model:**

"CDIF - Integrated Meta-model / Foundation Subject Area" [EIA111]

"CDIF - Integrated Meta-model / Common Subject Area" [EIA112]

"CDIF - Integrated Meta-model / Data Modeling Subject Area" [EIA114]

"CDIF - Integrated Meta-model / Data Flow Model Subject Area"

"CDIF - Integrated Meta-model / Data Definition Subject Area" *

"CDIF - Integrated Meta-model / State Event Model Subject Area" *

"CDIF - Integrated Meta-model / Physical Relational Database Subject Area" *

"CDIF - Integrated Meta-model / Presentation Location and Connectivity Subject Area"

"CDIF - Integrated Meta-model / Object-Oriented Analysis and Design Subject Area" *

"CDIF - Integrated Meta-model / Comp. Aided Control System Design Subject Area" *

"CDIF - Integrated Meta-model / Project Management Planning Subject Area" *

"CDIF - Integrated Meta-model / Business Process Modeling Subject Area" *

"CDIF - Integrated Meta-model / Expression Subject Area" *

**2.5.3 Detailed description of the subject areas**

This section contains information about some of the fundamental subject areas. It is not my intention to cover all of them, since most of them represent different modeling techniques. CDIF defines “data content” without specifying a syntax or API which would severely restrict the benefits of a standard. The different subject areas can be divided in two categories; the semantic information and the diagrammatic representation of models. A semantic concept can have zero or more graphical representations, or there could be models which are only in graphics. Furthermore, CDIF distinguishes what information to be transferred from the definition of how to transfer it. The information content of the Meta-model defines what information to transfer and the transfer format defines how the information must be structured, in order to complete a successful transfer. The Foundation SA is used to define all the other subject areas and must be used in a CDIF transfer. The
Background and foundation

Common SA is used to define concepts that are shared between many modeling techniques and the Data definition SA contains the information needed to define data. This subject area is referenced by the other subject areas when defining the data content of any metaobject.

Figure 8 presents a partition of the CDIF Integrated Meta-model, i.e. the Foundation SA (RootObject, RootEntity and RootEntity.isRelatedTo.RootEntity) and the Common SA (All the other meta-entities and meta-relationships in the figure). The Meta-model is expressed in the same EER variant as used in CDIF to specify the integrated Meta-model. An example describes how it is read: “A RootEntity may use an alternate name and it can use at most one (and only one) alternate name”. It is interesting to note that this way of specifying the cardinality constraints is the opposite to most ER variants, e.g. the Elmasri/Navathe variant [Elm94].

The three Meta-entities RootObject, RootEntity, and the Meta-relationship RootEntity.IsRelatedTo.RootEntity are abstract objects and provide the basis for the inheritance hierarchy of the integrated Meta-model. RootObject is the root object to the CDIF Attributable Meta Object Hierarchy (the IS_A hierarchy in the Meta-model). It holds the Meta-attributes CDIFIdentifier, DATECreated, DATEUpdated, TIMECreated and TIMEUpdated. Since it is the root object, every Meta-entity and Meta-relationship inherit these Meta-attributes. New Meta-attributes can be added using extensibility, however it is not possible to specify a supertype of this object. Of the Meta-attributes stated above, only the CDIFIdentifier is mandatory and need to be unique whenever it is used.

The relationship between the data modeling subject area and the common and foundation subject areas is that all Meta-entities found in the data modeling SA are subtypes of Semantic InformationObject. Hence, all meta-entities inherits the meta-attributes of RootObject, RootEntity and SemanticInformationObject. The same applies to other modeling techniques expressed in other subject areas.

A small example of how the Meta-model for the data modeling SA is used is presented in section 4.1.1 (p. 60).
The two most important subtypes of RootEntity is `SemanticInformationObject` and `PresentationInformationObject`. The `SemanticInformationObject` describes objects in a modeling technique, e.g. objects in a data model or business model. Examples of objects in a data model are entity, attribute and key. So for instance, the objects in the Meta-model of the data modeling SA are all subtypes of `SemanticInformationObject` and therefore, also inherits its Meta-attributes. These Meta-attributes are `BriefDescription` and `FullDescription`. The `PresentationInformationObject` describes graphical information about models, e.g. cordinates and shape of objects. It is also described in a separate SA. CDIF is a vast standard with many standard documents and in order to obtain more detailed information about the architecture we refer to the CDIF website [CDIF97a]. There is also support for defining types and domains within the data definition subject area. CDIF does not aim at covering every modeling technique in the software development process. Instead it is possible to extend the Meta-model with the aid of the subject area framework for modeling.
and extensibility. With the help of extensibility it is possible to define a modeling tech-
nique or create an extension of a subject area. As an example, consider extending the tradi-
tional dataflow diagram with timing information. It is possible to use the concepts from
the dataflow subject area and in addition create a new subject area, where the timing con-
structs are defined.

2.5.4 The architecture

The architecture of the Meta-model is a formal framework that defines the semantics of
different modeling languages. It is represented in a variant of an extended ER model with
the ability to form powerful inheritance hierarchies. It is complemented with rules to
extend the Meta-model, and the transfer format. The CDIF Integrated Meta-model con-
sists of a four level architecture similar to the generic architecture of Meta-models for
repositories. This architecture is presented in table 1:

<table>
<thead>
<tr>
<th>Meta-meta-model</th>
<th>e.g. “MetaEntity” and “MetaRelationship”</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meta-model</td>
<td>e.g. “Entity” and “Attribute”</td>
</tr>
<tr>
<td>Model</td>
<td>e.g. “Employee” and “Amount”</td>
</tr>
<tr>
<td>Data</td>
<td>e.g. “Susan” and “1250”</td>
</tr>
</tbody>
</table>

Table 1: The architecture of the Meta-model

The Data level constitutes the production database. The Model defines a UoD taken from
reality, e.g. an order and invoice system in a company. It is the Meta-model that defines the
standardized SA, i.e. the modeling languages. For example, an entity in an ER model can
have attributes or that a flow in a dataflow diagram connects to a process. Now, if one
wants to extend or change the semantics of a modeling language he would have to address
the Meta-meta-model in order to change the semantics of the Meta-model. These additions
or modifications must be expressed in a new SA guided by the rules of the extensibility
document. Imagine the addition of timing information into a dataflow model. The Meta-
meta-model would have to be extended so the Meta-model can express timestamps associ-
ated with flows.
2.5.5 The transfer format

The Transfer format CDIF provides support for various syntaxes and encodings. The syntax is a set of rules that determines terminal symbols from any instance of the CDIF Meta-Meta-model and from any instance of a Meta-model (the actual model data). The syntax is defined in the released standard SYNTAX.1 with a Clear-Text encoding in the released standard ENCODING.1. A CDIF transfer file includes a Header, Meta-model definitions and Model definitions. The header states the identity of the exporting tool and additional practical information. The Meta-model section states the type of model, e.g. data model or dataflowmodel together with extensions if necessary.

2.6 The database language SQL3

2.6.1 Active functionality

The demand for active functionality within DBS has increased in recent years, e.g. traffic control, cooperative problem solving, monitoring systems, secure banking applications etc. This section elaborates on the trigger mechanism used for active (reactive) functionality in the forthcoming database language SQL3. In order to elaborate on triggers we initially present the arguments for a centralized implementation of rules in database system.

Traditionally, database systems have been passive, i.e. they only execute queries which are submitted by users or application programs. New application demands have put a focus on active functionality within database systems. Active functionality can be defined as a database system that reacts to events without user intervention, i.e. the system acts like a peer and enforces specified rules. If one wants an information system to enforce active functionality there are several alternative ways of implementing this behaviour. The first alternative is to put the rules in the application programs. However, this involves complicated and extensive additions to the application code. Moreover, it can lead to inconsistencies between rules and they are also difficult to maintain due to the fact that the rules are scattered around the application code. The second alternative is to create a separate application program that polls the database in specified intervals to enforce the rules. The problem with this approach is the interval of the polling. If the application program polls the database to seldom it can result in rules not being enforced and if the application pro-
Background and foundation

gram polls the database to often it can result in overloading, greatly reducing the performance of the database system. Lastly, rules can be placed in the database which then enforces the active behaviour. Specifying a rulebase in the database system makes the rules easy to maintain, avoids inconsistencies and guarantees that rules always triggers. It also allows for multiple application programs to share the rules and optimization can be performed by the database system. Even though this way of implementing active functionality seem to be superior to the above mentioned ways, there are still issues waiting to be resolved, e.g. performance issues.

The rules that are used to specify active behaviour are structured according to the ECA mechanism. ECA stands for Event-Condition-Action and were introduced by Dayal [Day89], however their origin was established in the field of artificial intelligence in the mid eighties. Back then, the term used was “production rules” and encompassed a condition and an action part. The semantics of an ECA rule is “when an event occurs, check the condition and if it holds execute the action” [Wid95]. Note that sometimes rules lack the condition part. Examples of events are database events (e.g. insert and delete operations), external events (i.e. events that occur outside the database system) and temporal events. Conditions are boolean functions of data values and they do not change database states. Actions are data manipulation statements or external calls to application programs.

Most of the commercial relational database management systems of today support active functionality, e.g. in the form of triggers. Their capabilities are limited compared to prototypes found in the research community (which includes features such as composite events and coupling modes) [Act96]. In spite of this, triggers have had a great impact on commercial systems and will be standardized in the forthcoming SQL3 standard [ISO96]. Widom et al. identifies the need for a standardization of triggers [Wid95]:

“They lack standardization (active capabilities in commercial systems). Consequently, the various products have a wide variance in both the syntax and execution behavior of triggers. This results in a lack of uniformity, and the inability to use trigger applications on differing products.” (p. 234).

Triggers received attention in the process of defining SQL-92 but the technology was not mature enough and it was hard to reach consensus about the concept, so it was agreed that
triggers should be standardized in the forthcoming SQL3. This section defines the syntax of triggers proposed in the forthcoming database language SQL3. However, the standardization process is ongoing and there are bound to be some changes made. In spite of this many vendors have already specified their triggers according to the standard. The database vendors usually have a representative in the ongoing work of ANSI-ISO.

As mentioned above triggers follow the ECA mechanism; an event is a database operation, the condition is an arbitrary SQL predicate and the action is a sequence of SQL procedure statements.

### 2.6.2 Trigger syntax

The syntax of a trigger specified according to the database language SQL3 is presented below. The syntax is adopted from an ISO-ANSI working draft [ISO96].

```
<trigger definition> ::= 
    CREATE TRIGGER <trigger name>
    <trigger action time> <trigger event>
    ON <table name>
    [ REFERENCING <old or new values alias list> ]
    <triggered action>

<trigger action time> ::= 
    BEFORE 
    | AFTER

<trigger event> ::= 
    INSERT 
    | DELETE 
    | UPDATE [ OF <trigger column name list> ]

<trigger column list> ::=<column name list>

<triggered action> ::= 
    [ FOR EACH {ROW | STATEMENT } ]
    [ WHEN <left paren> search condition> <right paren> ]
    <triggered SQL statement>

<triggered SQL statement> ::= 
    <SQL procedure statement>
```
BEGIN ATOMIC
{ <SQL procedure statement> <semicolon> ]...
END

<old or new values alias> ::= 
  OLD [ AS ] <old values correlation name>
  | NEW [AS] <new values correlation name>
  | OLD_TABLE [ AS ] < old values table alias>
  | NEW_TABLE [ AS ] <new values table alias>

<old values table alias> ::= <identifier>
<new values table alias> ::= <identifier>
<old values correlation name> ::= <correlation name>
<new values correlation name> ::= <correlation name>

2.6.3 Trigger semantics

This section provides the semantics of triggers in SQL3. In addition the standard specifies a number of general rules for the syntax, e.g. if a <column name> is specified, then no <column name> shall appear more then once in the <trigger column list>. However, this section only specifies the core semantics. The more detailed rules are only elaborated on when necessary in section 5.1; defining a mapping procedure from CDIF to SQL3.

Every trigger must have a trigger name specified (<trigger name>). A trigger action can be specified as BEFORE or AFTER the triggering event (<triggering action time>). This allows for the specification of powerful rejection strategies. The trigger event (<trigger event>) can be specified as an INSERT, a DELETE or an UPDATE of a column name list. Every trigger must also have a specified table that it acts upon (<table name>) and it must be a base table.

A trigger may execute after every tuple that has been modified (FOR EACH ROW) or after the whole SQL statement (FOR EACH STATEMENT). The default for the trigger execution is FOR EACH STATEMENT. SQL3 makes it possible to reference the old or new value of a column or an entire table with respect to the triggering operation, by the referencing clause (REFERENCING <old or new values alias list>). Note that with three different triggering operations (INSERT, DELETE, UPDATE), two trigger action times
(BEFORE, AFTER) and two modes of granularity (STATEMENT, ROW) SQL3 therefore provide twelve distinct types of triggers available for a table.

The condition is an arbitrary SQL predicate (WHEN...) e.g. a boolean function or a query (nonempty results = TRUE) and the action part (<triggered SQL statement>) is a sequence of SQL statements.
3 Choice of model

3.1 Requirements of a model

One approach for expressing business rules is in a formal language. There are a wide variety of formal languages for describing rules. However, these are not always appropriate since they are too formal. There is a need for models that express business rules. A business rule approach to information systems development is a relatively new phenomenon and many of the common modeling techniques are insufficient for expressing rules, i.e. they do not provide a systematic treatment of business rules [Her94] [Ros94]. So, what are the requirements for a business rule model? Ross argues that there are four main characteristics that a model must possess [Ros94]. In his view a business rule model should be:

- **Expressive**: A business rule model must have a graphical representation and it should also be conceptual by nature. As we mentioned before (p. 20) conceptual models offer a number of advantages and are frequently used in software engineering.

- **Executable**: It should be possible to produce executable specifications (code) from the model. The objective of this characteristic is to avoid ambiguity and inconsistencies.

- **Extensive**: A rule model must be able to capture a wide variety of business rules types.

- **Extensible**: It should be possible to specify compound rules.

In addition Herbst et al. argue that there are some more specific characteristics that a business rule model must possess [Her94]:

- **Explicit rule or component support**: A rule model should either incorporate explicit semantic business rule constructs or it must express rules through its components, i.e. events, conditions and actions. A business rule model should support at least one of these alternatives.
Choice of model

- **Associations**: It should be possible to specify how business rules affect objects. Furthermore, dependencies between rules should be made explicit.

As stated above many of the traditional models used in software engineering are insufficient for expressing business rules. In particular, Entity-Relationship models and Dataflow models are restricted when used to express rules [Her94]. These two modeling techniques are the very foundation of structured analysis [Her94] and are closely connected to database design. ER models are appropriate for expressing some business rules as definitions and facts, together with some constraints. Examples of such rules are “an order must be related to a customer” or “a department may reside at multiple locations”. However, some business rules such as constraints and derivations are not possible to define in ER models. Examples of such rules are “no orders are accepted unless the customer has a sufficient credit rating” or “no orders that are less than five years old should be deleted”.

If the ER model is analysed according to the ECA [Day89] components, it is concluded by Herbst et al. that there is no explicit support for either of the three components [Her94]. Implicitly, the condition component is supported by cardinalities, keys and domains and the action component is partly supported by integrity maintenance (REJECT). If we consider Dataflow models, they have limited ability to express definitions and facts, however it is not possible to specify any constraints on the facts. If the Dataflow model is analysed according to the ECA components, it is concluded by Herbst et al. [Her94] that events are partially supported by dataflows. However, a dataflow does not necessarily trigger a process, since Dataflow diagrams does not support sequences of actions. The action component corresponds to processes but there is no support for the specification of conditions, i.e. only simple EA type business rules can be expressed in a Dataflow diagram. An example of such a rule is “every order should be confirmed”. Clearly, this is why the standardized modeling techniques currently found in CDIF are insufficient to express business rules.
3.2 Choice of model

3.2.1 Candidate models

The author has found some possible candidate models in the literature. The criteria for a candidate model is that it fulfills most of the requirements stated above. The following models seem to be suitable for expressing business rules in a systematic way:

- Tempora (The Entity-Relationship-Time (ERT) model and the Conceptual-Rule-Language (CRL)) [The91].
- The Entity-Relationship model with events and rules (ER2) [Tne90].
- The ROSS methodology [Ros94].

The TEMPORA project addressed an explicit business rule focus and the alignment of business knowledge. The project developed an ERT model which includes explicit modeling of time. As a complement to the data model a Conceptual Rule Language was defined that either places constraints on the objects, derives new information or acts as action rules.

The Entity-Relationship model with events and rules was originally intended to model the behaviour of active databases on the conceptual modeling level, however it can also be used to model business rules [Her94]. Some examples are presented in section 3.5.5. It extends the basic ER model with the semantic constructs of events and rules in a diagrammatic form. The ER2 diagram is complemented with a language that defines the events and rules.

The ROSS methodology is also based on an Entity Relationship approach and it is possible to use any variant of a specific ER model. Ross argues that a modeling technique for business rules should be data based:

“Large scale enterprises are characterized by a multitude of users, each having objectives that not infrequently conflict. Concurrency and substantial query requirements are the givens. What all these users have in common is the database, and the need to record and share persistent results in standard form. This strongly suggests that the expression of business rules should be based on data”
Therefore, the Ross methodology is based on any ER model with additional constructs to express business rules in a diagrammatic form. The ROSS methodology is an extensive modeling technique with the ability to express a wide variety of rules. It is the opinion of the author that in addition, if business rule models are based on data the process of modeling rules is easy to integrate with the traditional database design process.

### 3.2.2 Selection

The evaluation presented in this section emphasize the requirements that a model does not fullfill, in the opinion of the author. The ERT model and the CRL is an interesting approach to capture and model business rules. However, the main contribution regards the explicit modeling of time (many business rules incorporate time) and the approach lack some of the characteristics stated above. The time semantics included in the ERT model increases the complexity and it can be difficult to use the model for inexperienced users. Furthermore, the rules are explicitly supported (WHEN-IF-THEN) in a formal language and hence lack a graphical representation. The CRL can express relationship between rules and data model objects, but not graphically. There is no support for dependencies between rules.

The ROSS methodology seems more interesting since it provides a diagrammatic representation of the semantic constructs used to express rules. However, this model also suffer from some drawbacks. It includes a large number of operators and activators that increase the complexity. Also, it does not permit the ability to express actions. In this author’s opinion the ER2 model represents the best choice of the candidate modeling techniques. It supports explicit rule and event constructs with a graphical representation. It is also possible to express how rules affect data model objects and dependencies between rules in a diagrammatic format. There is a small set of extensions to the diagrammatic representation (events, rules and directed arcs - see section 3.5.5) and hence the model can still serve as a good base for communication between users and analysts. Since the rule and event extensions does not affect the characteristics of the ER model, we choose to use one of the most semantically rich Extended ER variant of Elmasri et al. [Elm85]. This has the advantage that we can capture a large subset of the business rule in the traditional EER model.
3.3 The Entity-Relationship Model (ER)

In 1976 Peter Chen presented the paper “The Entity-Relationship Model: Toward a unified view of data” [Che76] in response to the deficits of the relational model. The ER model can be seen as an extension to the relational model. Even though the relational model had a great impact on database technology, it lacked high level modeling constructs close to human perception. The ER model is a high level conceptual model and it has become the most popular model in conceptual design of databases, due to its simplicity and clear graphical presentation [Yan87]. It is acknowledged that there are over fifty variants of the EER model in the literature [Hal94]. This work specifically elaborates on the basic ER model of Elmasri/Navathe [Elm94] and the extended version of Elmasri et al. [Elm85] which is one of the most semantically rich EER models (the motivation for this choice is further elaborated on in section 3.4.1). Section 3.3.1 elaborates on the constructs of the basic ER model and section 3.3.2 describes the extended concepts.

3.3.1 Semantic constructs

The first basic building block in an ER model is an “entity” which denotes an independent thing in the real world. An “entity” may be tangible or conceptual. Examples of entities are employee, department and project\(^3\). The model distinguishes between entity types and instances of an entity type, e.g. the set of employees is an entity type and the specific employee “Susan” could be represented as an instance of the entity type employee. The notion of entity type and entity will be used interchangeably in this text. When mapping an ER model to the relational model and/or a physical model the entity types become the scheme definition (intension) and the instances populate the database (extension). Weak entities will be elaborated on in the section that defines constraints in the ER model. An entity can be described by its “attributes” which defines the properties of the entity, e.g. an employee can have the attributes name, address, birthdate, age and Social Security number (SSn). There are several types of attributes: simple-composite, singelvalued-multivalued and stored-derived attributes. Simple attributes are atomic by nature, e.g. SSn whereas

---

3. The COMPANY example used for explaining the concepts of the EER model is taken from [Elm85]. It is expressed in diagrammatic form in figure 9.
composite attributes denote divisible attributes, e.g. address which could be divided into Zip code and street. The reason for modeling composite attributes is in situations where a subpart of the attribute and the whole attribute is referred to interchangeably. Commonly, attributes are singlevalued but there are cases when attributes possess multiple values at the same time, e.g. the attribute location of the entity department, meaning that a department can reside at multiple locations. Most attributes are stored attributes, however it is sometimes possible to derive attributes from stored attributes or related entities. Examples of such attributes are age which can be derived from birthdate and number_of_employees which can be derived by counting the number of employees taking part in a specific project. Attributes must have a specified data type or an associated domain of values. Furthermore, attributes may have a special value of NULL, which can have several meanings. The NULL value of an attribute may represent that it is not applicable for an instance of an entity type or it may be unknown and in addition it may be missing. This is referred to as the problem of semantic overloading of NULL in the literature. The second basic building block in an ER model is the “relationship” which defines a set of associations between entity types. A relationship relates an entity to another entity and can be binary, ternary or n-ary (the number of participating entities, also referred to as the degree of the relationship). A relationship may also be recursive. An entity that participates in a relationship is said to play a role in that relationship. An example of a relationship is Works_for which states that an employee works for a department. Attributes may be specified on relationships which specifies that the attribute depends on the participating entities.

### 3.3.2 Constraints

It is possible to specify a variety of constraints in the ER model. One such constraint is the notion of a key attribute of an entity. This specifies that the attribute acts as a unique identifier for the entity. The natural key for the entity employee in the example is SSn, e.g. we can uniquely identify “Susan” if we know her SSn. Entities can have more then one key attribute. Another type of constraint is the one possible to define on domains of attributes. This makes it possible to define a value range on a domain, e.g. that the age of the employees should be between 16 and 70. A third type of constraint is defined on relationships. This states the cardinality and participation constraints. The cardinality ratio specifies the number of relationship instances that an entity can participate in. Possible cardinality
ratios are 1:1, 1:N and M:N. An example is the relationship employee works_for department with the cardinality ratio 1:N. This means that a department can have many employees but an employee can only work for one department. The participation constraints states the existence of an entity depending of another entity. Participation can be total or partial. An example of a total participation constraint is that every employee must work for a department. This is distinguished from a partial constraint, e.g. only some employees manage a department. Taken together, the cardinality and participation constraints are said to express the structural constraints of the ER model. The structural constraints imply insertion and deletion rules on entity types, e.g. an employee can not be deleted if he/she works for a department at the time being. An entity may be weak, expressing the fact that it can not exist independently of another entity and therefore has no own identifying attribute. The relationship that relates a weak entity to an entity (identifying owner) is called identifying relationship. A weak entity must always have a total participation constraint. The weak entity usually has a partial key which together with the identifying key of the identifying owner uniquely describes the weak entity. If we assume that no dependents ever have the same name in the example, the dependents name would be the partial key.

3.3.3 The ER diagram

It is important to distinguish between the semantics of an ER model and its diagrammatic representation. There is not a one-to-one correspondence between semantic constructs in the model and its graphical presentation, e.g. data types and domains are not usually specified in an ER diagram and in most cases, neither are instances of entity types. An example of an EER diagram is presented in section 3.4.4. That diagram will also be used to show an example mapping to the CDIF data modeling subject area.

3.4 The Extended Entity-Relationship Model (EER)

Even though the ER model provided for better modeling capabilities compared to the relational model, it was soon recognized that some important abstraction mechanisms\(^4\) were
Choice of model

missing. The basic ER model was not semantically rich enough to model some domains, which lead to the development of Extended ER models (EER), including new powerful modeling constructs, e.g. generalization/specialization and the category concept. Note that the EER model incorporates the ER modeling constructs and in addition extends it with additional semantic constructs.

3.4.1 Which EER variant to use?

What is the motivation for the specific Elmasri et al. variant of the EER model considered in this work? The choice was motivated by the fact that this specific variant is one of the most semantically rich models. A semantically rich EER model is capable of expressing more business rules, e.g. that an instance of the entity type EMPLOYEE must be a secretary or a mechanic, not both (the disjoint/overlapping constraint). Unfortunately, concepts are not always used to represent the same meaning among various EER models. Consider as an example the concept of a category used in both IDEF1X [Spe97] and the Elmasri/Navathe EER models. In IDEF1X, a categorization relationships denotes the common known generalization/specialization relationship (IS-A). In Elmasri/Navathe the category concept denotes superclass/subclass relationships with more then one superclass were the subclass is denoted “category”. Table 2 compare the semantic constructs in some popular EER variants and the terminology is adopted from Elmasri/Navathe. The table is partly adapted from Griebel [Gri96] but it has been extended.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Weak entity</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>1:1 relationship</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>1:N relationship</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>N:M relationship</td>
<td>yes</td>
<td>yes</td>
<td>-</td>
</tr>
<tr>
<td>Mutual exclusive relationship</td>
<td>-</td>
<td>yes</td>
<td>-</td>
</tr>
<tr>
<td>Ternary relationships</td>
<td>yes</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Participation/cardinality</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Relationship roles</td>
<td>yes</td>
<td>-</td>
<td>yes</td>
</tr>
<tr>
<td>Identifying relationship</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

48
### 3.4.2 Extended semantic constructs

One important semantic construct is the notion of superclass and subclass. An example is that members of an entity type BIKE and of an entity type MOTORBIKE may be grouped as a VEHICLE. A new superclass VEHICLE is created by the abstraction process generalization, i.e. generalization is the process of defining a generalized entity type from the given entity types.

The reverse process of generalization is specialization, i.e. the process of defining a set of subclasses of an entity type. An example is that members of the EMPLOYEE entity type

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Relationship with attribute</td>
<td>yes</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Relationship between rel.ship</td>
<td>yes</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Recursive relationship</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Attribute</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Key</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Foreign key</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Candidate key</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Ins./del./modification rules</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Domain</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Composite attribute</td>
<td>yes</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Derived attribute</td>
<td>yes</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Multivalued attribute</td>
<td>yes</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Null attribute</td>
<td>yes</td>
<td>-</td>
<td>yes</td>
</tr>
<tr>
<td>Superclass/subclass (IS-A)</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Inherited attributes</td>
<td>yes</td>
<td>yes</td>
<td>-</td>
</tr>
<tr>
<td>Disjoint/overlapping</td>
<td>yes</td>
<td>yes</td>
<td>-</td>
</tr>
<tr>
<td>Total/partial</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Defining predicate</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Multiple inheritance</td>
<td>yes</td>
<td>yes</td>
<td>-</td>
</tr>
<tr>
<td>Specialization/gen. process</td>
<td>yes</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Parallel subtypesets</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Category</td>
<td>yes</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 2: Semantic constructs in different EER variants
can be further grouped into the \textsc{hourly employee} and \textsc{salaried employee} entity types. This results in the superclass \textsc{employee} and the subclasses \textsc{hourly employee} and \textsc{salaried employee} (see Figure 9 for an example). The relationship between a superclass and its subclasses is often referred to as an IS-A relationship. It is sometimes important to distinguish which process that have been used, generalization or specialization and this can also be expressed in an EER diagram [Elm94]. The both distinct processes correspond to Bottom-Up versus Top-down design and even if it is common that they result in the same EER diagram, it is sometimes useful to keep track of which process that has been used [Elm94].

An important aspect of the IS-A relationship is the attribute inheritance mechanism. This means that the subclasses inherits all the attributes of its superclass and in addition can have their own attributes. Furthermore, the subclasses inherits all relationships from its superclass. The IS-A relationship may contain a discriminating single valued attribute (defining predicate) that functions as a selector for the subclasses. The discriminating attribute in the example would be \textsc{salarytype}. Sometimes we need to express multiple inheritance, e.g. a school may have the entity types \textsc{employee} and \textsc{student} and in addition students that are also employees (entity type \textsc{student-employee}) and therefore this entity type inherit all the attributes from its superclasses.

Another important semantic construct is the concept of a category. This is a superclass/subclass relationship with more then one superclass. The superclasses denote distinct entity types and the subclass is called a category [Elm94]. As an example, consider the case where we have the two entity types \textsc{department} and \textsc{company}. An \textsc{owner} may be either a \textsc{department} or a \textsc{company}. This is represented in diagrammatic form in Figure 9. The category \textsc{owner} is a subclass of the union of the two superclasses. This is shown with the U symbol. The inheritance mechanism of attributes works rather differently here and is sometimes referred to as selective inheritance, because the category (the subclass) inherits attributes from one of its superclasses, not both (compare with multiple inheritance).
3.4.3 Additional constraints

There are some additional constraints that are needed in an EER model. One such constraint regards the IS_A relationship and states the participation of the superclass instances in the corresponding subclasses. The relationship may be disjoint or overlapping. The disjoint constraint specifies that a superclass instance may only participate in one of its subclasses. If the relationship is overlapping the superclass instance may participate in all of its subclasses. Another type of constraint on the IS_A relationship is the total and partial constraint. A IS_A relationship may be total, stating that a superclass instance must participate in the relationship. It may also be partial, stating that it is optional for the superclass instance to participate in the relationship. The total/partial constraints also applies to categories. The specified constraints imply special insertion and deletion rules, e.g. a delete operation on a superclass entity type automatically implies the deletion of corresponding subclass entities.
3.4.4 The EER diagram

Figure 9: The company example
3.5 The Entity-Relationship Rules Model (ER2)

3.5.1 Constructs for expressing rules and events

The ER2 model is one of the few models with explicit constructs for expressing rules. The original presentation of the model was made in [Tnc90], however this work elaborates on the version presented in [Tan92]. It is an orthogonal extension to the previously discussed Extended ER model, i.e. the additional ER objects (events and rules) does not affect the semantics of the basic ER model (entities and relationships).

The additional constructs introduce active behaviour into the model by relating events and rules to entities and their attributes. This makes it possible to express both structural and dynamic aspects of the modelled UoD. Figure 10 presents the ER2 Meta-model.

![Figure 10: The ER2 Meta-model](image-url)
The basic semantic extension to the EER model is a behaviour sentence which can be composed of many behaviour sentences. The definition of a behaviour sentence follow the ECA paradigm:

**Definition 4:** When an event occurs, it triggers the corresponding rule which evaluates a condition, if the condition evaluates to true an action list is executed.

The above definition include the new functionality of the model and the events and rules are also expressed in an ER2 diagram. This section elaborates on a detailed level of the subparts of a behaviour sentence, since the components together make up the operational semantics of the language. For a formal definition of the operational semantics of the language we refer to Tanaka [Tan92].

### 3.5.2 Events

The syntax for an event is presented in table 3.

<table>
<thead>
<tr>
<th>event</th>
<th>[ BEFORE ] event_id <code>:</code> event_type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[ AND [ BEFORE ] event_id <code>:</code> event_type ]*</td>
</tr>
<tr>
<td>event_id</td>
<td>identifier</td>
</tr>
<tr>
<td>event_type</td>
<td>database_event</td>
</tr>
<tr>
<td>database_event</td>
<td>[ attribute_name OF ] object_name</td>
</tr>
<tr>
<td></td>
<td>MODIFIED</td>
</tr>
<tr>
<td></td>
<td>object_name INSERTED</td>
</tr>
<tr>
<td></td>
<td>object_name DELETED</td>
</tr>
<tr>
<td></td>
<td>[ attribute_name OF ] object_name RETRIEVED</td>
</tr>
<tr>
<td>object_name</td>
<td>identifier</td>
</tr>
<tr>
<td>attribute_name</td>
<td>identifier</td>
</tr>
<tr>
<td>external_event</td>
<td>signal</td>
</tr>
<tr>
<td>system_event</td>
<td>signal</td>
</tr>
<tr>
<td>signal</td>
<td>signal_name [ <code>(parameter_list)</code> ]</td>
</tr>
<tr>
<td>signal_name</td>
<td>identifier</td>
</tr>
<tr>
<td>parameter_list</td>
<td>value [ ``,` value ]*</td>
</tr>
</tbody>
</table>

Table 3: The syntax for an event.
Events are instantaneous and are distinguished from actions. Actions take time and causes events to occur. An example of an action is “modify project” and an example of an event is “project modified”. An event may precede another event or it may be unrelated to other events, and the precedence may be causal or temporal. Events can carry information to the firing rule by event attributes.

There are three types of events: database events include any operation of a database object or its attribute, external events are typically generated by application programs and finally, system events are signals generated by the system environment, e.g. the operating system. Events can also be temporal and origin from one of the three types of events stated above.

Conceptually, events are specified immediately before they occur and attempted operations are distinguished from completed operations. This makes the rule language more flexible and capable of modeling rejections, i.e. the possibility to reject an operation before actually executing it. This is stated with the optional BEFORE attribute of the event definition.

One important aspect of events is the composition of events into complex events. This has been elaborated on in the literature and some proposed operators include conjunction, disjunction and sequence [Cha93]. The ER2 model only allow the operator conjunction (AND), by the specification of the optional attribute AND on the event definition.

However, the disjunction operator of composite events can be defined implicitly by stating a behaviour sentence for every primitive event, which fire the same list of actions. Event_id is the unique identifier for the event, an object_name is an entity type or relationship type and attribute_name is the name of an attribute of an object. Signals are implementation dependent and are commonly stored procedures invoked from application programs.

### 3.5.3 Rules

The syntax for a rule is presented in table 4.

<table>
<thead>
<tr>
<th>rule</th>
<th>rule_id [ { description} ] [ { priority_level} ] [ IF condition THEN ] action_list</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule_id</td>
<td>identifier</td>
</tr>
</tbody>
</table>
Choice of model

<table>
<thead>
<tr>
<th>description</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>priority_level</td>
<td>identifier</td>
</tr>
</tbody>
</table>

Table 4: The syntax for a rule

A rule_id provides a unique identifier for the rule and the description describes the rule in text format. Furthermore, an optional priority_level could be specified to determine the order of rule execution. A condition is a predicate on the database and its specification is optional, corresponding to an Event-Action rule.

In general any SQL statement is allowed but commonly the predicate consists of a query which have to return a nonempty answer in order to evaluate to TRUE or a comparison between attributes. An action_list is a sequence of actions (denoted by the * in the table below).

3.5.4 Actions

The syntax for an action is presented in table 5.

<table>
<thead>
<tr>
<th>action_list</th>
<th>action [ ‘,’ action ]*</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>database_action</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>database_action</td>
<td>db_action ‘(‘event_id’)’</td>
</tr>
<tr>
<td>db_action</td>
<td>INSERT_ENTITY object_name ‘(‘value_list’)’</td>
</tr>
<tr>
<td></td>
<td>INSERT_RELATIONSHIP object_name [ ‘(‘value_list’’)’ ] BETWEEN rel_obj_list</td>
</tr>
<tr>
<td></td>
<td>DELETE_ENTITY object_name ‘(‘predicate’)’</td>
</tr>
<tr>
<td></td>
<td>DELETE_RELATIONSHIP object_name ‘(‘predicate’)’</td>
</tr>
<tr>
<td></td>
<td>DELETE_RELATIONSHIP object_name ‘(‘predicate’)’ BETWEEN rel_obj_list</td>
</tr>
<tr>
<td></td>
<td>MODIFY object_name ‘(‘predicate’)’</td>
</tr>
<tr>
<td></td>
<td>SET ‘(‘value_list’’)’</td>
</tr>
</tbody>
</table>

| value_list | assignment [ ‘,’ assignment ]* |
Choice of model

They are typically database operations or external actions, e.g. raising an exception. They can also be specified as REJECT_OPERATION or PROPAGATE_OPERATION corresponding to the semantics of the EER model. The database actions origins from the ER scheme and defines possible operations on the database (DDL statements). The semantics of REJECT_OPERATION is as follows [Tan92]:

“If BEFORE was specified with the firing event, then the suspended operation that would cause the event is aborted; otherwise, the operation is rolled back” (p. 66).

The semantics of the PROPAGATE_OPERATION (e_l) [Tan92]:

“selectively propagate the operation that caused the firing event to the adjacent objects listed in (e_l)” (p. 66).

The following pre-conditions need to be satisfied:

- Each event_id must be valid
- Each database_event must be a valid database event
- Each database event must correspond to its event identifier in the event type descriptor.

| assignment             | attribute [ OF object ]
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>‘=’ VALUE</td>
</tr>
<tr>
<td>rel_obj_list</td>
<td>rel_obj_pred [ AND rel_obj_pred ]*</td>
</tr>
<tr>
<td>rel_obj_pred</td>
<td>object_name [ ‘(‘attribute_name ‘=’ value’)’ ]</td>
</tr>
<tr>
<td>external_action</td>
<td>RAISE event_id ‘:’ signal_name</td>
</tr>
<tr>
<td></td>
<td>[ ‘(‘ actual_parameter_list’)’ ]</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>actual_parameter_list</td>
<td>value [ ‘,’ value ]</td>
</tr>
<tr>
<td>msg</td>
<td>string</td>
</tr>
<tr>
<td>db_event_list</td>
<td>event_id ‘:’ database_event</td>
</tr>
<tr>
<td></td>
<td>[ ‘,’ event_id ‘:’ database_event ]*</td>
</tr>
</tbody>
</table>

Table 5: The syntax for an action
• Let $of$ be the name of the object in the firing event and $oi$ the name of the object in the database event $di$, then:

  if $of$ is an entity type, then each $oi$ must be either an entity type connected to/from $of$ by an “ID” or “IS-A” arc, or a relationship type which $of$ participates on.

  if $of$ is a relationship type, then each $oi$ must be a relationship or entity type that participates in $of$.

The execution semantics:

• Let $oe$ be the name of the occurred update operation in the firing event (i.e. the occurred event on $of$). Then the execution semantics of each propagation follows the semantics of the corresponding individual operation:

  if $oe$ = “INSERTED” then execute insert operation on entity or relationship.

  if $oe$ = “DELETED” the execute delete operation on entity or relationship

  if $oe$ = “MODIFY” then execute modify operation on entity or relationship.

3.5.5 The ER2 diagram

The traditional EER diagram is extended by events and rules, events are represented by circles and rules by parallelograms. Some example business rules are presented below together with their diagrammatic representation found in figure 11. Directed arcs represent connections between events and rules and between events and data modeling objects (Entity or Relationship).

Arcs directed from the data modeling object to the event represents the fact that an action (operation) on the data modeling object causes the event to occur which in turn fires a rule. Arcs directed from the event to the data modeling object represents the fact that a rule raises an action(s) and that action causes an event to occur. This event commonly performs an operation on a data modeling object.

An example of the semantic distinction between arcs directed differently is business rule number 1 below. Here, the event that triggers the business rule (r1) is the modification operation of the “Project” entity type (e1). Whereas the rule evaluates the condition and if
it holds raises the event (e2) that performs a delete operation of all temporary “Employees”. This distinction is an important feature of the ER2 model.

BR1: “When the project is modified and the new budget is lower then the old budget, delete all temporary employees”

BR2: ”When an employee is deleted and he/she is a engineer refuse the deletion”

BR3: ”When an employee is deleted and considered temporary, remove him from relevant projects, from the relevant department and remove his dependents.

Figure 11: An ER2 example model
4 Extending CDIF

4.1 The CDIF data modeling subject area

Figure 13 depicts the standardized Meta-model for the data modeling subject area in CDIF [EIA114]. The standard document only includes diagrammatic partitions of the Meta-model, e.g. one partition describe roles, roleplayers and constraints. I have chosen to present the Meta-model in whole, since it gives a better overview of the constructs used to express a data model. The Meta-model aims at covering most common EER variants and it is based on the philosophy of a “least common denominator”. For a complete definition of all the Meta-attributes, Meta-entities and Meta-relationships in figure 13 see the standard document [EIA114]. The Meta-model is used in section 4.2.2 which presents a description of mapping the chosen EER model to CDIF. Furthermore, the Meta-model is extended in section 4.4 to incorporate the event and rule extensions (found in the ER2 model) used to model business rules.

4.1.1 A small example

Figure 12 presents a simple ER scheme which states that “A customer may place zero or more orders” and “each order must be placed by one (and only one) customer”.

![Diagram](image-url)
In order to show the practical use of CDIF we present a description of mapping the information in the marked area (for spacesaving reasons) of the scheme into a CDIF transfer file. The presentation of the syntax is inspired of the transfer file. To follow which Meta-entities and Meta-relationships that are used in the data modeling SA of CDIF we refer to figure 13 (the Meta-model). No Meta-attributes are shown in the figure. It should also be noted that the CDIF Meta-attribute CDIFIdentifier is mandatory in every Meta-entity and Meta-relationship. In addition there are a number of inherited or local Meta-attributes that may apply, e.g. the local Meta-attribute “EntityType” of the Meta-entity “Entity”. For a complete definition of all possible Meta-attributes we refer to the standard document [EIA114].

1. We need to define an instance of the CDIF Meta-entity “Datamodel” for each Datamodel. This is represented in CDIF with the Meta-entity “Datamodel”:

   Meta-entity .......... Datamodel
   Meta-attributes
   CDIFIdentifier .......... DM01
   Name ........... “An example datamodel”
   BriefDescription ........ “Customer-order model”

2. The scheme contains the entity type “Customer” which is represented in CDIF with the Meta-entity “entity”. The optional Meta-attribute “EntityType” of “Entity” states that the entity has got an independent existence:

   Meta-entity .......... Entity
   Meta-attributes
   CDIFIdentifier .......... ENT01
   Name ........... “Customer”
   BriefDescription ........ “Our company has got customers”
   EntityType ... “Kernel”
3. We need to relate the “Customer” to the specific Datamodel which is represented in CDIF with the Meta-relationship “Datamodel.Collects.DataModel Objects”:

Meta-relationshipDataModel.Collects.DataModelObjects
   Meta-attributes
   CDIFIdentifier .......... DM_ICO_DMO01
   SourceEntity .............. DM01
   DestinationEntity........ ENT01

4. We need to specify the attribute “Name” which is represented in CDIF with the Meta-entity “Attribute”:

Meta-entity ............. Attribute
   Meta-attributes
   CDIFIdentifier .......... AT01
   Name ........... .............. “Name”
   BriefDescription......... “The name of a customer”

5. We need to specify that the attribute “Name” belongs to the entity type “Customer”. This is represented in CDIF with the Meta-relationship “DefinitionObject.Contains.ComponentObject”:

Meta-relationship..... DefinitionObject.Contains.ComponentObject
   Meta-attributes
   CDIFIdentifier .......... DO_C_CO01
   SourceEntity .............. ENT01
   DestinationEntity........ AT01
4.2 Mapping EER to CDIF

This section presents an analysis whether the chosen EER model, i.e. the Elmasri/Navathe variant [Elm94] (elaborated on in section 3.3 and 3.4) can be specified according to CDIF (table 6). The issue regards whether all the semantic constructs and constraints are possible to specify according to the CDIF repository standard (the data modeling subject area found in figure 13). In addition, this section also presents a description of the process of mapping an example EER scheme to CDIF. The objective of the example is twofold; firstly, it will demonstrate the usage of the standard for interchanging information between CASE-tools. Secondly, the example incorporates those semantic constructs that are not explicitly or implicitly supported in CDIF. This provides the basis for an elaboration of why the semantic constructs cannot be specified according to CDIF.

4.2.1 Semantic constructs and constraints

A distinction is made whether it is possible to represent the constructs in a straight forward fashion (explicit) or rather indirectly (implicit). Furthermore, note that the semantic constructs and constraints are specified according to the terminology of Elmasri/Navathe.

Table 6 presents the result of the analysis:

<table>
<thead>
<tr>
<th>EER [Elm94]</th>
<th>Explicit</th>
<th>CDIF Implicit</th>
<th>No support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Weak entity</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Relationship</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1:1 relationship</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1:N relationship</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M:N relationship</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N-ary relationship</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Participation/cardinality</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Relationship roles</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Identifying relationship</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Relationship with attribute</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Relations. between relations.</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Recursive relationship</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### 4.2.2 An example mapping

This section presents a description of mapping the example model in figure 14 to the data modeling subject area of CDIF [EIA114], i.e. it explains how the CDIF integrated Meta-model is used for representing the semantics of the EER model in figure 14. The CDIF subject areas that will be used are Foundation SA, Common SA and Data modeling SA. The ordering of the steps in the description is based on the structure of a CDIF transfer file. We have chosen to describe one example of mapping a semantic construct to CDIF. The complete mapping of the example model is presented in appendix A. For instance,
step number 2 in the mapping description only consider the entity type “Employee”. All other entity types are presented in appendix A.

Figure 14: An example model

1. The first meta-entity that needs to be defined is the datamodel of the example:

   Meta-entity .............. Datamodel
   Meta-attributes
   CDIFIdentifier....... MOD01
   Name..................... “Example”
   Brief Description... “An example Data Model”
   ModelType ............ “Conceptual”
   DateCreated.......... 08-06-1997
   TimeCreated......... 01:01:59

2. Next, every entity in the data model needs to be defined:

   Meta-entity .............. Entity
   Meta-attributes
   CDIFIdentifier....... ENT02
3. Every entity needs to be related to the data model. This is done with the Meta-relationship `Datamodel.Collects.DatamodelObject`. Note that entity is a DataModelObject:

   Meta-relationship `Datamodel.Collects.DatamodelObject`  
   Meta-attributes  
   CDIFIdentifier....... RELID02  
   SourceEntity........ MOD01  
   Destination ............ EntityENT02

4. The entity called Employee has got two subtypes. This is defined with the Meta-entity `SubtypeSet` and the Meta-relationships `InheritableDataModelObject.Is.SupertypeFor.SubtypeSet` and `InheritableDataModelObject.Is.SubtypeFor.SubtypeSet`. The subtyping is defined as disjoint, which means that an employee can not both be salaried and hourly compensated. This constraint need to be stated with the optional Meta-attribute `IsExclusive`:

   Meta-entity .......... SubtypeSet  
   Meta-attributes  
   CDIFIdentifier...... ENT09  
   Name .................. “Employee subtypes”  
   BriefDescription.... “This defines the subtyping of Employees”  
   IsExclusive .......... TRUE

   Meta-relationship..... InheritableDataModelObject.Is.Supertype.For.SubtypeSet  
   Meta-attributes  
   CDIFIdentifier...... RELID09  
   SourceEntity......... ENT02  
   DestinationEntity .. ENT09

   Meta-relationship...... InheritableDataModelObject.Is.Subtype.For.SubtypeSet  
   Meta-attributes  
   CDIFIdentifier...... RELID10  
   SourceEntity......... ENT03  
   DestinationEntity .. ENT09

   Meta-relationship...... InheritableDataModelObject.Is.SubtypeFor.SubtypeSet  
   Meta-attributes  
   CDIFIdentifier...... RELID11  
   SourceEntity......... ENT04  
   DestinationEntity .. ENT09
5. Every relationship in the example model needs to be defined.

Meta-entity ............... Relationship
Meta-attributes
CDIFIdentifier....... ENT10
Name .................. "Works_for"
BriefDescription.... "This relates an employee to a department, i.e who works on which department"

6. All relationships in the example need to be related to the data model.

Meta-relationship..... DataModel.Collects.DataModelObject
Meta-attributes
CDIFIdentifier....... RELID12
SourceEntity......... MOD01
DestinationEntity .. ENT10

7. Next, roles are defined and connected to relationships. Roleplayers are connected to their roles which in turn are connected to their entity. Note that the Elmasri/Navathe EER model only models "outer" cardinality values [EIA114, p. 24]. This is solved by assigning 1:1 to the "outer" cardinalities and mapping the values of the "outer" cardinalities in the model to the "inner" cardinalities. This is the way that it should be done according to the CDIF standard body:

Meta-entity ............... Role
Meta-attributes
CDIFIdentifier....... ROLE01
Name .................. "parentworksfor"
IsMaster............... TRUE
IsSource............... TRUE

Meta-entity ............... RolePlayer
Meta-attributes
CDIFIdentifier....... ROLP01
Name .................. "parentworksfor"
MinInnerCardinality ........... "0"
MaxInnerCardinality ........... "N"
MinOuterCardinality .......... "1"
MaxOuterCardinality ........... "1"

Meta-relationship..... Role.BelongsTo.Relationship
Meta-attributes
CDIFIdentifier....... ROL_BT_REL01
SourceEntity......... ROLE01
Extending CDIF

DestinationEntity ..ENT10

Meta-relationship.....Datamodelobject.ActsAs.RolePlayer
Meta-attributes
CDIFIdentifier.......DMO_AA_ROLP01
SourceEntity.........ENT06
DestinationEntity ..ROLP01

Meta-relationship.....RolePlayer.Plays.Role
Meta-attributes
CDIFIdentifier.......ROL_P_ROLE01
SourceEntity.........ROLP01
DestinationEntity ..ROLE01

Meta-entity ..............Role
Meta-attributes
CDIFIdentifier.......ROLE02
Name ...................“childworksfor”
IsMaster...............TRUE
IsSource...............TRUE

Meta-entity ..............RolePlayer
Meta-attributes
CDIFIdentifier.......ROLP02
Name ...................“childworksfor”
MinInnerCardinality ........“1“
MaxInnerCardinality ........“1“
MinOuterCardinality ......“1”
MaxOuterCardinality ......“1”

Meta-relationship.....Role.BelongsTo.Relationship
Meta-attributes
CDIFIdentifier.......ROL_BT_REL02
SourceEntity.........ROLE02
DestinationEntity ..ENT10

Meta-relationship.....Datamodelobject.ActsAs.RolePlayer
Meta-attributes
CDIFIdentifier.......DMO_AA_ROLP01
SourceEntity.........ENT02
DestinationEntity ..ROLP02

Meta-relationship.....RolePlayer.Plays.Role
Meta-attributes
CDIFIdentifier.......ROL_P_ROLE02
SourceEntity.........ROLP02
DestinationEntity ..ROLE02
8. The next step is to define all attributes. We start off with the attributes of the entity “Employee”:

Meta-entity .............. Attribute
Meta-attributes
CDIFIdentifier....... ATT01
Name .................... “SSn”
BriefDescription.... “The identifying social security number”

9. These attributes needs to be related to their entity:

Meta-relationship..... DefinitionObject.Contains.ComponentObject
Meta-attributes
CDIFIdentifier....... DO_C_CO01
SourceEntity.......... ENT02
DestinationEntity .. ATT01

10. The attributes for the entity “Salaried employee” needs to be defined as “inherited” from employee:

Meta-relationship..... Attribute.IsInheritedFrom.Attribute
Meta-attributes
CDIFIdentifier....... A_IIF_A01
SourceEntity.......... ATT05
DestinationEntity .. ATT01

11. The attribute “Numberof” needs to defined as derived:

Meta-entity .............. ProjectedAttribute
Meta-attributes
CDIFIdentifier....... ATT18
Name .................... “Nr_of_emp”
BriefDescription.... “The total number of employees on a department”

Meta-attributes
CDIFIdentifier....... PA_IPO_A01
SourceEntity.......... ATT18
DestinationEntityATT16

12. Next, we need to define the unique keys for the entities in the model. This is done with the Meta-entity Candidatekey and the Meta-relationships Entity.IsIdentified_by.Key and Key.Incorporates.Attribute.

Meta-entity .............. Candidatekey
Meta-attributes
4.2.3 Analysis of semantic constructs with no explicit support

Insertion and deletion rules for Supertype/subtype relationships: There are a number of insertion and deletion rules for supertype/subtype relationships, e.g. deleting an entity from the supertype automatically implies that it should be deleted from all subtypes that it belongs to. CDIF lack support for the specification of these specific rules.

Category: categories, i.e. subtypes with multiple supertypes and selective inheritance of attributes are not possible to specify according to CDIF. This relates to the fact that it is not possible to specify selective inheritance of attributes. All attributes must be inherited from the supertype to the subtype according to CDIF.

Weak entity: It is possible to specify that an “Entity” is of the type characteristic meaning that the entity type depends upon the existence of another entity. However the concept as defined by Elmasri/Navathe [Elm94] is not fully supported since there is no way to specify according to which entity the entity is weak. The one-to-one (1:1) cardinality/participation constraint, i.e. the Meta-attributes MinInnerCardinality etc. of the Meta-entity RolePlayer, simply does not rule out the possibility that the weak entity may be related to other relationships with 1:1 constraints.

Composite attributes: CDIF does not offer any support for the specification of composite attributes.

Multivalued attributes: CDIF does not offer any support for the specification of composite
attributes.

4.2.4 Discussion

There are some comments to be made regarding the semantic constructs above. Some might argue that there is no need for explicit semantic constructs for expressing composite attributes in a conceptual diagram, since it is possible to express this requirement with the attribute being specified as an entity, e.g. the entity “Department” is related to the entity type “Location” with the one-to-many (1:N) relationship “has”. However, it is highly questionable if this is a natural way to specify this requirement, i.e. if the resulting scheme can be seen as expressive and serve as a good base for communication among users. This doubt relates to the basic principles of the ER model stating that an entity is a thing or concept with the attributes being characteristics of that entity. Expressing attributes as entities because they have multiple values clearly violates these principles. We should also remember that conceptual models should be free from implementation details. Similarly, the same arguments apply to ER variants that do not allow for many-to-many (N:M) relationships.

4.2.5 Proposed additions

In order to correct the lack of semantic constructs above we make the following suggested additions to the Meta-model:

- Add a Meta-relationship `Attribute.IsComposedOf.Attribute` to allow the possibility to specify composite attributes.

- Add the Meta-entity `MultipleAttribute` and define it as a subtype of `Attribute`. This would allow for the specification of multiple attributes.

- Add the Meta-attribute `RelationshipType` to the Meta-entity `Relationship` to allow the possibility for explicitly specifying Identifying Relationships and weak entities.

- Add Meta-attributes to the Meta-relationship `InheritableDatModelObject.IsSubTypeIn.SubtypeSet` to model the behaviour rules governing update and deletion rules for IS-A relationships.
The concept of a category is not straightforward to incorporate into CDIF. It must be emphasized that the category concept is very different from multiple inheritance when regarding the semantics. So, a multiple inheritance situation with selective attribute inheritance is not a sufficient specification of a category. The standard explicitly states that selective inheritance of attributes is not supported. However, the category concept is not very common and we do not propose any extensions to incorporate this construct into the standard.

### 4.3 Rules for extending CDIF

The rules for extending the CDIF integrated Meta-model is defined in the subject area “Framework for Modeling and Extensibility” [EIA107]. The document also includes the definition of the integrated Meta-meta-model. The extensibility mechanism is an important feature of the CDIF Meta-model which makes it possible to exchange information between tools that is not defined in the standardized Meta-model. This section presents the core rules for using extensibility in a transfer. For a detailed description of the extensibility mechanism see the standard document. The detailed level rules that are affected are further elaborated on in the description of the mapping procedure from ER2 to CDIF. Basically, there are a number of additions possible to define on the CDIF integrated Meta-model (any of the standardized SA for a modeling technique):

- Add new subject area.
  
  It is possible to define a new subject area. It may define new Meta-entities, Meta-relationships or Meta-attributes or use existing ones.

- Add CollectableMetaObject.IsUsedIn.SubjectArea.
  
  All Meta-entities that are explicitly used in the new subject area must be defined with this Meta-relationship.

- Add Meta-entity.

- Add Meta-relationship.
Extending CDIF

  It is possible to define additional supertypes for existing meta-entities or Meta-relationships.
- Add Meta-attribute to Meta-entity or Meta-relationship.
- Add Legal Enumerated Value to a Meta-attribute.

4.4 The semantic extensions

This section elaborates on the extended semantic constructs that have been added to the data modeling SA of the CDIF integrated Meta-model [EIA114]. All Meta-entities and Meta-relationships are given a unique identifier with the Meta-attribute CDIFIdentifier. The Meta-model is presented in diagrammatic form in section 4.5. All extensions have been designed to enable a one-to-one mapping from the ER2 model to the CDIF representation (see figure 15).

4.4.1 Events

Events are represented by the Meta-entity “event”. An event may be either a database event, a system event or an external event. This is specified with the Meta-attribute “EventType” of the Meta-entity “Event”. Events may theoretically be specified before they occur and this is defined with the Meta-attribute “EventOccurrence” of “event”. The name of an external event is specified with the Meta-attribute “ExternalEvent” and possible parameters are specified with the Meta-attribute “ExternalParameters” which is a list of parameters. The name of a system event is specified with the Meta-attribute “SystemEvent” and possible parameters are specified with the Meta-attribute “SystemParameters” which is also a list. An event may fire a rule which is specified with the Meta-relationship “Event.Fires.Rule”. An event must fire a rule and in addition it may fire several rules which is represented with a one-to-many (1:N) relationship. An event may precede another event which is specified with the Meta-relationship “Event.Precedes.Event”. An event may precede another event or it may precede several events which is represented with a zero-to-many (0:N) relationship.
An event may affect an entity or a relationship which is specified with the Meta-relation-ship “Event.Affects.InheritableDatModelObject”. Entities and relationships are subtypes of “InheritableDataModelObjects. An example is that an action (which is a part of a rule) raises an event which in turn affects an entity, e.g. the event deletes an entity instance. An event may affect many entities or relationships which is specified with a zero-to-many (0:N) relationship. In addition an entity or relationship may be affected by the occurrence of an event meaning that the action on the entity or relationship is the event. This is specified with the Meta-relationship “InheritableDataModelObject.Affected_by.Event. The operations possible on the two Meta-relationships above is insert or delete. This is specified with the Meta-attribute “DatabaseEvent” on the above Meta-relationships. Events may also affect attributes or be affected by attributes. This is specified similar to the above Meta-relationships, i.e. “Event.Affects.Attribute” or “Attribute.Affected_by.Event and the Meta-attribute “DatabaseEvent” which can be a modification or a retrieve. Events may also be composed of other events which is specified with the Meta-relationship “Event.Is_Composed_Of.Event” with the Meta-attribute “Operator” that defines the conjunction AND. In addition events may contain attributes which is specified with the Meta-relationship “ComponentObject.Contains.DefinitionObject”

4.4.2 Rules
Rules are represented with the Meta-entity “Rule”. A rule may have a priority level which is specified with the Meta-attribute “PriorityLevel”. A rule must contain one action and in addition it can contain several actions which is specified with the Meta-relationship “Rule.Contains.Action. The Meta-relationship has got the cardinality/participation constraints 1:N.

4.4.3 Actions
Actions are represented with the Meta-entity “Action”. An action may be a database, external, reject or propagate action. This is specified with the Meta-attribute “Action-Type”. An action may raise one or more events which is specified with the Meta-relationship “Action.Raises.Event” and which Event is specified with Event_Identifier. In addition actions may contain attributes which is specified with the Meta-relationship “Componen-
tObject.Contains.DefinitionObject”. The type of action is specified with the Meta-attribute DB_Action, e.g. Insert Entity.
4.5 The extended Meta-model

Figure 15: The extended Meta-model
4.6 The detailed definition of the Meta-model

A new subject area must be defined:

Name: ER2
CDIFMetaIdentifier: NEW00001
Description: This SA defines the ER2 model
VersionNumber: 01.00

4.6.1 Meta-Entities

The new MetaEntity “Event” must be defined:

MetaEntity: ME001
Name: Event
Subtype of: DataModelObject
Description: “An Event fires a rule and theoretically, has no
Extending CDIF

Type

Inherited Meta-Attributes

CDIFIdentifier from RootObject
DateCreated from RootObject
DateUpdated from RootObject
TimeCreated from RootObject
TimeUpdated from RootObject
BriefDescription from SemanticInformationObject
FullDescription from SemanticInformationObject
Name from DefinitionObject
Operator from DefinitionObject
SpecificationLanguage from DefinitionObject
SpecificationText from DefinitionObject

Local Meta-Attributes

EventType
EventOccurrence
ExternalEvent
ExternalParameters
SystemEvent
SystemParameters

Inherited Meta-Relationships

DataModelObject.IsMemberOf.datamodelsubset from DMObject
Cluster.Collects.DataModelObject from DMObject
DataModel.Collects.DataModelObject from DMObject
RolePlayer.RefinesForSubtype.DataModelObject from DMObject
DataModelObjectActsAs.RolePlayer from DMObject
DefinitionObject.Contains.ComponentObject from DefinitionObject
DefinitionObject.IsConstructedWith.ProjectionComponent from DefinitionObject
ProjectionComponent.IsFullProjectionOf.DefinitionObject from DefinitionObject

Local Meta-Relationships

Event.Affects.InheritableDataModelObject
InheritableDataModelObject.Affected_by.Event
Event.Affects.Attribute
Attribute.Affected_by.Event
Event.Precedes.Event
Event.Fires.Rule
Rule.Raises.Event
Extending CDIF

Event.IsComposedOf.Event

<table>
<thead>
<tr>
<th>META-ATTRIBUTE NAME</th>
<th>EventType^a</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDIFMetaIdentifier</td>
<td>MA01</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>This defines the type of event</td>
</tr>
<tr>
<td>USAGE</td>
<td></td>
</tr>
<tr>
<td>ALIASES</td>
<td></td>
</tr>
<tr>
<td>CONSTRAINTS</td>
<td></td>
</tr>
<tr>
<td>DATATYPE</td>
<td>Enumerated</td>
</tr>
<tr>
<td>DOMAIN</td>
<td>Database_event, External_event, System_event</td>
</tr>
<tr>
<td>LENGTH</td>
<td></td>
</tr>
<tr>
<td>ISOPTIONAL</td>
<td>False</td>
</tr>
</tbody>
</table>

^a. This table includes all information that is possible to specify on a Meta-attribute according to CDIF even if there is no information available, e.g. USAGE. This is avoided hereafter for space-saving reasons.

<table>
<thead>
<tr>
<th>META-ATTRIBUTE NAME</th>
<th>EventOccurrence</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDIFMetaIdentifier</td>
<td>MA02</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>This defines a point in time before the event has occurred</td>
</tr>
<tr>
<td>DATATYPE</td>
<td>Enumerated</td>
</tr>
<tr>
<td>DOMAIN</td>
<td>Before, After</td>
</tr>
<tr>
<td>ISOPTIONAL</td>
<td>True</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>META-ATTRIBUTE NAME</th>
<th>ExternalEvent</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDIFMetaIdentifier</td>
<td>MA05</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>This defines the type of event</td>
</tr>
<tr>
<td>DATATYPE</td>
<td>STRING 256</td>
</tr>
<tr>
<td>ISOOPTIONAL</td>
<td>True</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>META-ATTRIBUTE NAME</th>
<th>ExternalParameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDIFMetaIdentifier</td>
<td>MA04</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>The parameters that are passed from the event</td>
</tr>
<tr>
<td>DATATYPE</td>
<td>IntegerList</td>
</tr>
<tr>
<td>CONSTRAINT</td>
<td>This Meta-attribute may only be instantiated if ExternalEvent is instantiated.</td>
</tr>
<tr>
<td>ISOOPTIONAL</td>
<td>True</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>META-ATTRIBUTE NAME</th>
<th>SystemEvent</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDIFMetaIdentifier</td>
<td>MA05</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>This defines the type of event</td>
</tr>
<tr>
<td>DATATYPE</td>
<td>STRING 256</td>
</tr>
</tbody>
</table>
An event is a subtype of DataModelObject:
AttributableMetaObject.HasSubtype.AttributableMetaObject
SourceEntity 1012
DestinationEntity ME001
CollectableMetaObject.IsUsedIn.SubjectArea
SourceEntity ME001
DestinationEntity NEW00001

The new MetaEntity “Rule” must be defined:

MetaEntity ME002
Name Rule
SubtypeOf DataModelObject
Description “An event fires a rule”
Type Kernel

Inherited Meta-Attributes

CDIFIdentifier from RootObject
DateCreated from RootObject
DateUpdated from RootObject
TimeCreated from RootObject
TimeUpdated from RootObject
BriefDescription from SemanticInformationObject
FullDescription from SemanticInformationObject
Name from DefinitionObject
Operator from DefinitionObject
SpecificationLanguage from DefinitionObject
SpecificationText from DefinitionObject

Local Meta-Attributes

PriorityLevel
Inherited Meta-Relationship

DefinitionObject.IsConstructedWith.ProjectionComponent from DefinitionObject
ProjectionComponent.IsFullProjectionOf.DefinitionObject from DefinitionObject
DataModelObject.IsMemberOf.Datamodelsubset from DMObject
Cluster.Collects.DataModelObject from DMObject
DataModel.Collects.DataModelObject from DMObject
RolePlayer.RefinesForSubtype.DataModelObject from DMObject
DataModelObject.ActsAs.RolePlayer from DMObject
DefinitionObject.Contains.ComponentObject from DefinitionObject

local Meta-Relationships

Event.Fires.Rule
Rule.Contains.Action

<table>
<thead>
<tr>
<th>META-ATTRIBUTE NAME</th>
<th>PriorityLevel</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDIFMETAIDENTIFIER</td>
<td>MA07</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>This userprovided value defines priority between rules</td>
</tr>
<tr>
<td>DATATYPE</td>
<td>Integer</td>
</tr>
<tr>
<td>DOMAIN</td>
<td>Positive integers</td>
</tr>
<tr>
<td>ISOPTIONAL</td>
<td>True</td>
</tr>
</tbody>
</table>

AttributableMetaObject.HasSubtype.AttributableMetaObject

SourceEntity 1012
DestinationEntity ME002

CollectableMetaObject.IsUsedIn.SubjectArea

SourceEntity ME002
DestinationEntity NEW00001

The new MetaEntity “Action” must be defined:

MetaEntity ME003
Name Action
SubtypeOf SemanticInformationObject
Description “A Rule contains an Action”
Type Kernel

Inherited Meta-Attributes

CDIFIdentifier from RootObject
DateCreated from RootObject
### Extending CDIF

<table>
<thead>
<tr>
<th>Local Meta-Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DateUpdated</strong></td>
</tr>
<tr>
<td><strong>TimeCreated</strong></td>
</tr>
<tr>
<td><strong>TimeUpdated</strong></td>
</tr>
<tr>
<td><strong>BriefDescription</strong></td>
</tr>
<tr>
<td><strong>FullDescription</strong></td>
</tr>
<tr>
<td><strong>Name</strong></td>
</tr>
<tr>
<td><strong>Operator</strong></td>
</tr>
<tr>
<td><strong>SpecificationLanguage</strong></td>
</tr>
<tr>
<td><strong>SpecificationText</strong></td>
</tr>
</tbody>
</table>

**Inherited Meta-Relationships**

- DefinitionObject.Contains.ComponentObject from DefinitionObject
- DefinitionObject.IsConstructedWith.ProjectionComponent from DefinitionObject
- ProjectionComponent.IsFullProjectionOf.DefinitionObject from DefinitionObject

**Local Meta-Relationships**

- Rule.Contains.Action
- Action.Raises.Event

<table>
<thead>
<tr>
<th>META-ATTRIBUTE NAME</th>
<th>ActionType</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDIFMETAIDENTIFIER</td>
<td>MA08</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>The type of action</td>
</tr>
<tr>
<td>DATATYPE</td>
<td>Enumerated</td>
</tr>
<tr>
<td>DOMAIN</td>
<td>Database_action, External_action, REJECT, PROPAGATE</td>
</tr>
<tr>
<td>ISOPTIONAL</td>
<td>True/False</td>
</tr>
</tbody>
</table>

**AttributableMetaObject.HasSubtype.AttributableMetaObject**

- SourceEntity 3
- DestinationEntity ME003

**CollectableMetaObject.IsUsedIn.SubjectArea**

- SourceEntity ME003
- DestinationEntity NEW00001

#### 4.6.2 Meta-Relationships

- MetaRelationship MR001
- Name InheritableDataModelObject.Affected_by.Event
**Extending CDIF**

Description     “This relates an Event to an Entity or Relationship”
MinSourceCard   "0"
MaxSourceCard   "N"
MinDestCard     "0"
MaxDestCard     "N"

**Inherited Meta-Attributes**

* 5

**Local Meta-Attributes**

Database_Event

<table>
<thead>
<tr>
<th>META-ATTRIBUTE NAME</th>
<th>Database_Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDIFMETAIDENTIFIER</td>
<td>MA09</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>This defines the type of database event</td>
</tr>
<tr>
<td>DATATYPE</td>
<td>Enumerated</td>
</tr>
<tr>
<td>DOMAIN</td>
<td>Inserted, Deleted</td>
</tr>
<tr>
<td>ISOPTIONAL</td>
<td>False</td>
</tr>
</tbody>
</table>

MetaRelationship.HasSource.MetaEntity

<table>
<thead>
<tr>
<th>SourceEntity</th>
<th>MR001</th>
</tr>
</thead>
<tbody>
<tr>
<td>DestinationEntity</td>
<td>1033</td>
</tr>
</tbody>
</table>

MetaRelationship.HasDestination.MetaEntity

<table>
<thead>
<tr>
<th>SourceEntity</th>
<th>MR001</th>
</tr>
</thead>
<tbody>
<tr>
<td>DestinationEntity</td>
<td>ME001</td>
</tr>
</tbody>
</table>

AttributableMetaObject.HasSubtype.AttributableMetaObject

<table>
<thead>
<tr>
<th>SourceEntity</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>DestinationEntity</td>
<td>MR001</td>
</tr>
</tbody>
</table>

MetaRelationship

<table>
<thead>
<tr>
<th>Name</th>
<th>Event.Affects.InheritableDataModelObject.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>“This relates an Event to an Entity or Relationship”</td>
</tr>
<tr>
<td>MinSourceCard</td>
<td>&quot;0&quot;</td>
</tr>
</tbody>
</table>

5. The inherited Meta-Attributes from RootObject and SemanticInformationObject, e.g. CDIFIdentifier,...... (see the Meta-Entity “Event” for a complete description. These are suppressed to keep the document from growing to large.

84
Extending CDIF

MaxSourceCard "N"
MinDestCard "0"
MaxDestCard "N"

*Inherited Meta-Attributes*

\* 

*Local Meta-Attributes*

Database_Event
Predefined_table_NEW
Predefined_table_OLD

<table>
<thead>
<tr>
<th>META-ATTRIBUTE NAME</th>
<th>Database_Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDIFMETAIDENTIFIER</td>
<td>MA10</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>This defines the type of database event</td>
</tr>
<tr>
<td>DATATYPE</td>
<td>STRING</td>
</tr>
<tr>
<td>DOMAIN</td>
<td>Inserted, Deleted</td>
</tr>
<tr>
<td>ISOPTIONAL</td>
<td>False</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>META-ATTRIBUTE NAME</th>
<th>Predefined_table_NEW</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDIFMETAIDENTIFIER</td>
<td>MA11</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>The event carries alias for new table</td>
</tr>
<tr>
<td>DATATYPE</td>
<td>Identifier</td>
</tr>
<tr>
<td>DOMAIN</td>
<td>A valid Entity identifier</td>
</tr>
<tr>
<td>CONSTRAINTS</td>
<td>Must be specified if Database_Event=Inserted</td>
</tr>
<tr>
<td>ISOPTIONAL</td>
<td>False/True</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>META-ATTRIBUTE NAME</th>
<th>Predefined_table_OLD</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDIFMETAIDENTIFIER</td>
<td>MA12</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>The event carries alias for old table</td>
</tr>
<tr>
<td>DATATYPE</td>
<td>Identifier</td>
</tr>
<tr>
<td>DOMAIN</td>
<td>A valid Entity identifier</td>
</tr>
<tr>
<td>CONSTRAINTS</td>
<td>Must be specified if Database_Event=Deleted</td>
</tr>
<tr>
<td>ISOOPTIONAL</td>
<td>False/True</td>
</tr>
</tbody>
</table>

MetaRelationship.HasSource.MetaEntity

SourceEntity MR002
DestinationEntity ME001

MetaRelationship.HasDestination.MetaEntity
### Extending CDIF

<table>
<thead>
<tr>
<th>SourceEntity</th>
<th>MR002</th>
</tr>
</thead>
<tbody>
<tr>
<td>DestinationEntity</td>
<td>1033</td>
</tr>
</tbody>
</table>

**AttributableMetaObject.HasSubtype.AttributableMetaObject**

<table>
<thead>
<tr>
<th>SourceEntity</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>DestinationEntity</td>
<td>MR002</td>
</tr>
<tr>
<td>MetaRelationship</td>
<td>MR003</td>
</tr>
<tr>
<td>Name</td>
<td>Event.Precedes.Event</td>
</tr>
<tr>
<td>Description</td>
<td>This relates an Event to an Event</td>
</tr>
<tr>
<td>MinSourceCard</td>
<td>&quot;0&quot;</td>
</tr>
<tr>
<td>MaxSourceCard</td>
<td>&quot;N&quot;</td>
</tr>
<tr>
<td>MinDestCard</td>
<td>&quot;0&quot;</td>
</tr>
<tr>
<td>MaxDestCard</td>
<td>&quot;N&quot;</td>
</tr>
</tbody>
</table>

**Inherited Meta-Attributes**

*  

**Local Meta-Attributes**

**MetaRelationship.HasSource.MetaEntity**

<table>
<thead>
<tr>
<th>SourceEntity</th>
<th>MR003</th>
</tr>
</thead>
<tbody>
<tr>
<td>DestinationEntity</td>
<td>ME001</td>
</tr>
</tbody>
</table>

**MetaRelationship.HasDestination.MetaEntity**

<table>
<thead>
<tr>
<th>SourceEntity</th>
<th>MR003</th>
</tr>
</thead>
<tbody>
<tr>
<td>DestinationEntity</td>
<td>ME001</td>
</tr>
</tbody>
</table>

**AttributableMetaObject.HasSubtype.AttributableMetaObject**

<table>
<thead>
<tr>
<th>SourceEntity</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>DestinationEntity</td>
<td>MR003</td>
</tr>
<tr>
<td>MetaRelationship</td>
<td>MR004</td>
</tr>
<tr>
<td>Name</td>
<td>Event.Fires.Rule</td>
</tr>
<tr>
<td>Description</td>
<td>This relates an Event to a Rule.</td>
</tr>
<tr>
<td>MinSourceCard</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>MaxSourceCard</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>MinDestCard</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>MaxDestCard</td>
<td>&quot;N&quot;</td>
</tr>
</tbody>
</table>

**Inherited Meta-Attributes**

86
Extending CDIF

*  

Local Meta-Attributes

MetaRelationship.HasSource.MetaEntity

SourceEntity  MR004
DestinationEntity  ME001

MetaRelationship.HasDestination.MetaEntity

SourceEntity  MR004
DestinationEntity  ME002

AttributableMetaObject.HasSubtype.AttributableMetaObject

SourceEntity  3
DestinationEntity  MR004

MetaRelationship  MR006
Name  Attribute.Affected_by.Event
Description  This relates an Event to an Attribute
MinSourceCard  "0"
MaxSourceCard  "N"
MinDestCard  "0"
MaxDestCard  "N"

Inherited Meta-Attributes

*

Local Meta-Attributes

Database_Event

MetaRelationship.HasSource.MetaEntity

SourceEntity  MR006
DestinationEntity  17

MetaRelationship.HasDestination.MetaEntity

SourceEntity  MR006
DestinationEntity  ME001

AttributableMetaObject.HasSubtype.AttributableMetaObject

SourceEntity  3
Extending CDIF

DestinationEntity MR006

MetaRelationship MR007
Name Event.Affects.Attribute
Description This relates an Event to an
MinSourceCard "0"
MaxSourceCard "N"
MinDestCard "0"
MaxDestCard "N"

Inherited Meta-Attributes *

Local Meta-Attributes

Database_Event
Predefined_Attribute_NEW
Predefined_Attribute_OLD

META-ATTRIBUTE NAME Database_Event
CDIFMETAIDENTIFIER MA13
DESCRIPTION This defines the type of database event
DATATYPE STRING
DOMAIN Modified, Retrieved
ISOPTIONAL False

META-ATTRIBUTE NAME Predefined_Attribute_NEW
CDIFMETAIDENTIFIER MA14
DESCRIPTION The event carries alias for new attribute name
DATATYPE STRING
ISOPTIONAL False

META-ATTRIBUTE NAME Predefined_Attribute_OLD
CDIFMETAIDENTIFIER MA15
DESCRIPTION The event carries alias for old attribute name
DATATYPE STRING
ISOPTIONAL False

MetaRelationship.HasSource.MetaEntity

SourceEntity MR007
DestinationEntity ME001

MetaRelationship.HasDestination.MetaEntity
Extending CDIF

SourceEntity: MR007
DestinationEntity: 17

AttributableMetaObject.HasSubtype.AttributableMetaObject

SourceEntity: 3
DestinationEntity: MR007

MetaRelationship: MR008
Name: Event.Is_Composed_Of.Event
Description: This defines composite Events
MinSourceCard: "0"
MaxSourceCard: "N"
MinDestCard: "0"
MaxDestCard: "N"

Inherited Meta-Attributes

*

Local Meta-Attributes

Operator

<table>
<thead>
<tr>
<th>META-ATTRIBUTE NAME</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDIFMETAIDENTIFIER</td>
<td>MA16</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>This operator defines a conjunction between events</td>
</tr>
<tr>
<td>DATATYPE</td>
<td>Enumerated</td>
</tr>
<tr>
<td>DOMAINS</td>
<td>AND</td>
</tr>
<tr>
<td>ISOPTIONAL</td>
<td>False</td>
</tr>
</tbody>
</table>

MetaRelationship.HasSource.MetaEntity

SourceEntity: MR008
DestinationEntity: ME001

MetaRelationship.HasDestination.MetaEntity

SourceEntity: MR008
DestinationEntity: ME001

AttributableMetaObject.HasSubtype.AttributableMetaObject

SourceEntity: 3
DestinationEntity: MR008
**Extending CDIF**

<table>
<thead>
<tr>
<th>MetaRelationship</th>
<th>MR009</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Rule.Contains.Action</td>
</tr>
<tr>
<td>Description</td>
<td>This relates actions to rules</td>
</tr>
<tr>
<td>MinSourceCard</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>MaxSourceCard</td>
<td>&quot;N&quot;</td>
</tr>
<tr>
<td>MinDestCard</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>MaxDestCard</td>
<td>&quot;N&quot;</td>
</tr>
</tbody>
</table>

**Inherited Meta-Attributes**

*

**Local Meta-Attributes**

MetaRelationship.HasSource.MetaEntity

<table>
<thead>
<tr>
<th>SourceEntity</th>
<th>MR009</th>
</tr>
</thead>
<tbody>
<tr>
<td>DestinationEntity</td>
<td>ME002</td>
</tr>
</tbody>
</table>

MetaRelationship.HasDestination.MetaEntity

<table>
<thead>
<tr>
<th>SourceEntity</th>
<th>MR009</th>
</tr>
</thead>
<tbody>
<tr>
<td>DestinationEntity</td>
<td>ME003</td>
</tr>
</tbody>
</table>

AttributableMetaObject.HasSubtype AttributableMetaObject

<table>
<thead>
<tr>
<th>SourceEntity</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>DestinationEntity</td>
<td>MR009</td>
</tr>
</tbody>
</table>

MetaRelationship | MR010 |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Action.Raises.Event</td>
</tr>
<tr>
<td>Description</td>
<td>This states what actions produce events</td>
</tr>
<tr>
<td>MinSourceCard</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>MaxSourceCard</td>
<td>&quot;N&quot;</td>
</tr>
<tr>
<td>MinDestCard</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>MaxDestCard</td>
<td>&quot;N&quot;</td>
</tr>
</tbody>
</table>

**Inherited Meta-Attributes**

*

**Local Meta-Attributes**

DBAction
EventId
ValueList
Predicate
NameOfSignal
Extending CDIF

ExternalAction
Msg

**META-ATTRIBUTE NAME** | **DBAction**
---|---
**CDIFMetaIdentifier** | MA17
**DataType** | Enumerated
**Domain** | INSERT_ENTITY, INSERT_RELATIONSHIP, DELETE_ENTITY, DELETE_RELATIONSHIP, DELETE_RELATIONSHIP_BETWEEN, MODIFY_SET
ISOPTIONAL | True

**META-ATTRIBUTE NAME** | **EventId**
---|---
**CDIFMetaIdentifier** | MA18
**DataType** | Identifier
ISOPTIONAL | True

**META-ATTRIBUTE NAME** | **ValueList**
---|---
**CDIFMetaIdentifier** | MA19
**DataType** | TEXT
ISOPTIONAL | False

**META-ATTRIBUTE NAME** | **Predicate**
---|---
**CDIFMetaIdentifier** | MA20
**DataType** | TEXT
ISOPTIONAL | True

**META-ATTRIBUTE NAME** | **NameOfSignal**
---|---
**CDIFMetaIdentifier** | MA21
**DataType** | TEXT
ISOPTIONAL | True

**META-ATTRIBUTE NAME** | **ExternalAction**
---|---
**CDIFMetaIdentifier** | MA22
**DataType** | Enumerated
**Domain** | RAISE, MESSAGE
ISOPTIONAL | True

**META-ATTRIBUTE NAME** | **Msg**
---|---
**CDIFMetaIdentifier** | MA23
**DataType** | TEXT
ISOPTIONAL | False
MetaRelationship.HasSource.MetaEntity
SourceEntity       MR010
DestinationEntity  ME003

MetaRelationship.HasDestination.MetaEntity
SourceEntity       MR010
DestinationEntity  1033

AttributableMetaObject.HasSubtype.AttributableMetaObject
SourceEntity       3
DestinationEntity  MR010

4.7 Evaluation of the design

This section comments on the design of the extended Subject Area for the ER2 model defined above. Events and Rules are specified as subtypes of the Meta-entity Inheritable-DataModelObject. This design choice is motivated by the fact that they are “first class modeling objects” together with Entities and Relationships. However, this design choice raises a problem since Rules and Events inherit all the Meta-relationships of its supertype in the Meta-model. Some of these Meta-relationships apply to Events and Rules and are very useful, e.g. the ability to specify subsets of a data model with the inherited Meta-relationship DataModelObject.IsMemberOf.DataModelSubset. This Meta-relationship provides the ability to group Events and Rules together with Entities and Relationships in a subset of a Data model. However, some of the inherited Meta-relationships does not apply to Events and Rules, e.g. the Meta-relationship DataModelObject.ActsAs.Roleplayer. This is not considered to be a major problem since it is optional to instantiate Meta-relationships in a transfer according to CDIF.

One might also raise the question about the specification of conditions in the design. The condition part of a Rule is defined using SQL language with the constraint that the SQL statements must reference Entities, Relationships or Attributes specified in the instance model. The problem with this approach is that the SQL statements specify operations on a Entity-Relationship model instead of a relational model.
4.8 Mapping ER2 to CDIF

This section presents a description of mapping the ER2 model to CDIF, i.e. the mapping procedure regards the rule and event extensions to the EER model. The description of the mapping procedure uses the new subject area for the ER2 model defined in appendix B and in addition references the standardized data modeling subject area.

4.8.1 The data model

In spite of the fact that the description of the mapping have been divided into two parts, they should be viewed as a whole. This means that the Meta-entity `Datamodel` should not be instantiated again (conceptually we are dealing with one model expressing business rules). Note that it is mandatory to apply the Meta-attribute `CDIFIdentifier` to every instance of a Meta-entity or Meta-relationship. The Meta-attributes that are mandatory will be identified, all other inherited or local Meta-attributes are optional.

4.8.2 Events

1. Define one instance of the Meta-entity `event` for every event in the model. There are a number of optional inherited and local Meta-attributes of the Meta-entity `event`. Define an instance of every optional Meta-attribute necessary, e.g. `DateCreated` or `BriefDescription`. Define an appropriate name for the event and specify the mandatory Meta-attribute `EventType` (`Database_event`, `external_event` or `system_event`).

2. If the event is of type `Database_event`: specify one of the following alternatives:

   If the event affects an EER object (entity or relationship) define one instance of the Meta-relationship `Event.Affects.InheritableDataModelObject` and add the Meta-attribute `Database_event` which states the type of operation. Specify if the operation is of type `Inserted` or `Deleted`. If there is a need to reference new or old values of the attributes of the Entity specify the Meta-attribute `Predefined_table_NEW` or `Predefined_table_OLD`.

   If the EER object is affected by the event define one instance of the Meta-relationship `InheritableDataModelObject.Affected_by.Event` and specify
the type of operation (Inserted or Deleted).

If the event affects the attribute of an EER object define one instance of the Meta-relationship \textit{Event.Affects.Attribute} and specify the type of operation (\textit{Retrieve} or \textit{Modify}). If there is a need to reference new or old values of the attributes of the Entity specify the Meta-attribute \textit{Predefined\_attribute\_NEW} or \textit{Predefined\_attribute\_OLD}.

If the attribute of an EER object is affected by an event define one instance of the Meta-relationship \textit{Attribute.Affected\_by.Event} and specify the type of operation (\textit{Retrieve} or \textit{Modify}).

3. If the event is of type external add the Meta-attribute \textit{EventType} = External and add the Meta-attribute \textit{ExternalParameters} that specify parameters if needed.

4. If the event is of type system add the Meta-attribute \textit{Event\_Type} = System and add system parameters if needed.

5. If there is a need to specify the event immediately before it occurs (to model reject operations) add the Meta-attribute \textit{EventOccurrence} with the value \textit{BEFORE}.

6. If the event is composed of other events define one instance of the Meta-relationship \textit{Event.Is\_Composed\_of\_Event} and add the Meta-attribute \textit{Operator} = \textit{AND}.

\textbf{4.8.3 Rules}

1. Define one instance of the Meta-entity \textit{rule} for every rule in the model.

2. Add the optional Meta-attributes needed, e.g. \textit{name} and \textit{priority} level.

3. For every event that fires a rule define one instance of the Meta-relationship \textit{Event\_Fires\_Rule}.

4. If the rule has got a condition part add the Meta-attribute \textit{Specification\_Language} = SQL. Add the Meta-attribute \textit{Specification\_Text} which must be a SQL predicate or a query. Also, the \textit{Specification\_Text} must only reference valid constructs from the model, i.e. valid names of entities or attributes.
4.8.4 Action

1. Define one instance of the Meta-entity action for every action that belongs to a rule.

2. Add the Meta-attribute ActionType (Database, External, REJECT, PROPAGATE).

3. Define one instance of the Meta-relationship Rule.Contains.action for every action that belongs to a rule.


5. Add the Meta-attribute DB_Action (Insert Entity, Insert Relationship, Delete entity, Delete Relationship, Delete Relationship between or Modify_Set.

6. Add the Meta-attribute Event_Identifier which states the uniqueidentifier of the Event.

7. Add the Meta-attribute Valuelist.

8. Add the Meta-attribute Predicate.

9. Add the Meta-attribute NameOfSignal.

10. Add the Meta-attribute External_Action.

11. Add the Meta-attribute Msg.

4.8.5 Discussion

The above section shows that it is possible to keep all semantic information of the rule and event extensions of the ER2 model in the mapping procedure. This may come as no surprise since this was the main objective when designing the extended Meta-model.
5 Mapping to SQL3

5.1 Introduction

This section presents a description of mapping the CDIF representation into triggers in SQL3. The description is to be considered generic, i.e. every step must be instantiated for every semantic construct in CDIF. An example is the first step which should be instantiated for every Meta-attribute Name of the Meta-entity Rule. Comments and problems with the description of the mapping procedure can be found in the next section.

5.2 The mapping description

1. Define one instance of the <trigger name>. The information shall originate from the Meta-attribute Name of the Meta-entity Rule.

2. Define one instance of the <table name>. The information shall originate from the Meta-attribute Name of the InheritableDataModelObject that is related to the rule via the Event. The two relationships stating this fact are Event.Affects.InheritableDataModelObject and Event.Fires.Rule.

3. Define one instance of the <trigger event>. The information about which event fires which rule is obtained from the Meta-relationship Event.Fires.Rule. The information about which event affects which entity, relationship or attributes of an entity is obtained from the Meta-relationships below. Specify one of the following alternatives:

   If there exist an instance of the Meta-relationship Event.Affects.InheritableDataModelObject, define one instance of the <trigger event>. The information shall originate from the Meta-attribute Database_Event of the Meta-relationship Event.Affects.InheritableDataModelObject. This Meta-attribute can have the value INSERT or DELETE which is mapped to the corresponding INSERT and DELETE of the <trigger event>.

   If there exist an instance of the Meta-relationship Event.AffectsAttribute, define one instance of the <trigger event> and <trigger column name>...
list>. The information shall originate from the Meta-attribute Database_Event of the Meta-relationship Event.Affects.Attribute. This Meta-attribute can have the value MODIFY or RETRIEVE which is mapped to the corresponding UPDATE of the <trigger event>. The information that determines the <trigger column name list> origins from the instances of the Meta-relationship Event.Affects.Attribute.

4. Define one instance of the <trigger action time>. The information shall origin from the Meta-attribute EventOccurence of the Meta-entity Event. The Meta-attribute can have the value BEFORE or AFTER which is mapped to the corresponding BEFORE or AFTER of the <trigger actiontime>.

5. If a Rule has got a condition part, i.e. if there is information contained in the Meta-attributes SpecificationLanguage and SpecificationText of the Meta-entity Rule, then specify the search condition of the <triggered action>. The information shall originate from the Meta-attribute SpecificationText.

6. If an Attribute is referenced using OLD and NEW values, i.e. if there is information contained in the Meta-attributes Predfined_Attribute_OLD and Predefined_Attribute_NEW of the Meta-relationship Event.Affects.Attribute, then map these to their corresponding <old values correlation name> and <new values correlation name> in the trigger REFERENCING clause.

7. If an Entity is referenced using OLD and NEW values, i.e. if there is information contained in the Meta-attributes Predfined_Table_OLD or Predefined_Table_NEW of the Meta-relationship Event.Affects.InheritableDataModelObject, then map these to their corresponding <old values table alias> or <new values table alias> in the trigger REFERENCING clause.

5.2.1 Discussion

The mapping procedure does not consider stored procedures in SQL3, hence only Events and Conditions are being mapped to their corresponding trigger definition statements, i.e. all the Meta-attributes of the Meta-entity Action is not attended to here. The Action part is left for future work, however it should be straightforward to complete the mapping.
Mapping to SQL3

description to incorporate the whole ER2 model. In addition the mapping procedure discards the rules that govern the execution of the triggers, e.g. if <old values table alias> or <new values table alias> is specified then FOR EACH STATEMENT shall be specified. These detailed rules are found in the SQL3 standard and are not further elaborated on here.

There is a mismatch of semantic constructs between the ER2 model (represented by the CDIF constructs above) and triggers in SQL3. For example, the ER2 model allow for the concept of composite events, something which is not found in triggers and vice versa, e.g. SQL3 allow for the concept of tuple or statement granularity of a trigger. This section elaborates on these deficits of the mapping procedure. The question raised here, regards how useful such a mapping procedure might be considering these deficits. It is the opinion of this author that the mapping procedure still might be very useful to a developer.

Assume that we have a tool that implements the above mapping procedure. In response to the constructs available in the ER2 model with no corresponding constructs in triggers, the tool could support suggestions for which rules found in the conceptual model to be transformed into triggers. Note that all rules are not suitable to specify as centralized database rules and some are even impossible to specify, e.g. in the case where have composite events which is not possible to specify according to SQL3 or any commercial system today. However, the design information attained at the conceptual modeling level could still be used to aid the development of application programs. In response to the additional constructs found in triggers but not in the ER2 model the tool could implement a user dialogue or use defaults, e.g. when considering the granularity of a trigger.

In the ER2 model an event can be of the types; database event, system event (e.g. a call from the operating system) or external event (e.g. a call from an application program). SQL3 only consider database events, hence this information can not be specified according to SQL3. This means that the Meta-attributes SystemEvent and ExternalEvent can not be mapped to SQL3. Also, the ER2 model can specify composite events which is represented by the Meta-relationship Event.IsComposedOf.Event in CDIF. This information is not possible to specify according to SQL3 since triggers only consider atomic events. The ER2 model makes a distinction between a modify or retrieve operation of attributes. SQL3
only permits the specification of an update operation of a column, hence we lose this semantic information. The reason for this might be that very few rules are triggered by retrieve operations. SQL3 allow for the specification of the granularity of a trigger (FOR EACH ROW or FOR EACH STATEMENT). There are no such corresponding constructs to map from the CDIF representation since the ER2 model does not make this distinction. The ER2 model permits the specification of a priority level of a rule which determines the order of rule execution. Priority levels are not possible to specify on triggers. This is a good example of semantic information that could be defined as an implementation-dependent definition.
6 Discussion and conclusions

6.1 Introduction

This chapter presents the achievements of this work. The following subsections correspond to the three main objectives from the introduction. In addition, section 6.5 presents some reflections of the CDIF standard which has been a major focus in this thesis.

It is important to note the fundamental view underlying the conclusions in this chapter. The first comment regards the view taken of the CDIF standard in this work. As mentioned in the introduction the view taken here is the one of a transfer format, where the Meta-model serves as the basis for the production of a transfer file. This implies that any comment regarding how effective the CDIF Meta-model is for specifying a repository scheme is not considered to be of interest. The second comment regards the important distinction between semantic and presentation information specified in CDIF. This work strictly concentrates on the semantics of a modeling technique and hence no conclusions elaborate on features that relate to presentation information. It is also important to note that the foundation for the semantics of the chosen model in this work originates in the literature, not in a vendor dependent implementation of a model.

6.2 Choice of model

It is the opinion of the author that the ER2 model is well suited to serve the purpose of promoting communication between users and analysts. In addition, it must be concluded that the model can specify a wide range of business rule types, e.g. high-level policy business rule and enforcement of integrity constraints. We believe this is a consequence of the support of the E-C-A components of a rule in the model. There are some comments to be made regarding the choice of modeling technique for expressing Business Rules in this work.

The first comment regards the number of candidate models involved in the selection process. It can be argued that the number of candidate models is rather small. This is explained by the fact that very few modeling techniques provide explicit semantic constructs for
Discussion and conclusions

rules and even fewer models allow these rule constructs to be expressed in a diagrammatic form. This work emphasize the importance of expressive modeling techniques as a basis for communication between users and analysts, hence the low number of candidate models. In addition, the creators of the ER2 modeling approach noted a major drawback concerning the specification of rules at the type/instances level.

“The approach lacked the ability to specify rules to instances rather then only to types” [Cha97].

One might argue that the rule and event extensions to an EER model increases the complexity and hence the model is no longer suited for communicating between users and analysts. Indeed, by incorporating more constructs the complexity in the model increases. However, it is my strong belief that the complexity issue does not constitute a major problem. One way of solving this problem is presented in section 6.5 containing some overall reflections.

6.3 Extending CDIF

6.3.1 Mapping EER to CDIF

For a Case Data Interchange Format standard to gain wide acceptance, it is fundamental that it supports a wide variety of data modeling variants. This work shows that the Data Modeling Subject Area of CDIF can incorporate one of the most semantically rich EER variants reasonably well.

In spite of the lack of support for composite attributes, multiple attributes, weak entities, identifying relationships, the category concept and update and deletion rules for IS_A relationships, it must be concluded that the Integrated Meta-model is well suited for exchanging most EER variants. This is especially true if we consider the fact that the category concept is not a common feature in most EER models. The concept of a category with the selective inheritance mechanism of attributes can be simulated simply by excluding the Meta-relationship \texttt{Attribute.IsInheritedFrom.Attribute} for some attributes. This Meta-relationship populates the attributes of the subtype entity. The problem with this approach is that the standard clearly states that all attributes must be inherited in a supertype/subtype relationship, hence this approach does not adhere to the standard.
Furthermore, we have highlighted that multiple attributes can be expressed as entities in their own rights. However, we did raise the question of whether the resulting EER scheme can be seen as expressive and serve the purpose to promote communication between people. Specifying a multiple attribute as an entity also raises the question of whether it rather should be an aspect of a standardized SA for physical data modeling (currently in working group review by CDIF). The same argument applies to the standardized concept of an AccessPath found in the Data Modeling SA (further elaborated on in the section of general comments of CDIF). Probably, the most serious deficit regards the lack of semantic constructs for describing the rules governing the instances of supertype/subtype relationships. We find this somewhat odd since it is possible to specify the dynamic behaviour rules associated with referential integrity.

An alternative to choosing a semantically rich EER variant from the literature is to use a model implemented in a tool as the basis for analysis. The problem with this approach is the risk of confusing the semantics from the model with implementation-dependent features of the tool. This results in a conflict with the philosophy of CDIF as “a least common denominator” among various modeling techniques, not tool features.

Based on the experience of working with and analyzing the data modeling SA of CDIF we conclude that the quality of the Meta-model is good.

We address the lack of some semantic constructs in the data modeling Subject Area by proposing minor additions to the Meta-model. We also argue that the concept of a category should not be included into the Meta-model.

### 6.3.2 Extending the Meta-model

In addition to the issue of support for various variants of a modeling technique, it is also crucial that a standard can be extended, i.e. the Meta-model of CDIF can be extended to incorporate information which is vendor specific. This work shows that the extensibility feature of CDIF is a powerful and easy to use mechanism. It is powerful because it can express most modeling techniques. This relates to the additions possible to the Meta-model, e.g. add Meta-entity, add Meta-relationship and add Meta-attribute. These operations can express the semantics of most models, since they are by nature very general. The extensibility mechanism is usable since it basically constitutes an Extended Entity Rela-
Discussion and conclusions

This work specifically show that it is possible to create a hybrid model using the extensibility mechanism, i.e. by adding semantic Meta-constructs to incorporate the rule and event extensions found in the ER2 model into the standardized data modeling SA. All extensions are specified in a new subject area with references to the standardized SA. Probably, this implies that it is possible to specify completely new modeling techniques using the extensibility mechanism, since there are no existing Meta-model semantics to take into account, i.e. that task would considered to be easier. Of course, this is also relative to the complexity of the modeling technique.

There are some minor problems in the design of the new subject area for the ER2 model, e.g. the specification of the Meta-entities \textit{Event} and \textit{Rule} as subtypes of \textit{InheritableDataModelObject}. \textit{Event} and \textit{Rule} inherit all the Meta-relationships of \textit{InheritableDataModelObject} and some of these simply do not apply to an \textit{Event} or a \textit{Rule}. This is not considered to be a major problem since it is optional to instantiate Meta-relationships in a transfer according to CDIF.

6.3.3 Mapping ER2 to CDIF

The mapping procedure of mapping the rule and event extension found in the ER2 model according to the extended Meta-model is straightforward. This is not a surprise since the ultimate goal of designing the new subject area was to incorporate the semantics of the ER2 model. One requirement of the design is that it would only allow for a one-to-one mapping of constructs and this requirement has been fulfilled. This relates to the fact that we have shown the possibility to map each individual semantic construct in a one-to-one fashion.
6.4 Mapping to SQL3

We show that it is possible to map the CDIF representations to triggers in SQL3. The mapping procedure suffers from some deficits, e.g. the exclusion of stored procedures and the mismatch of semantic constructs between ER2 and triggers. However, we conclude that the mapping procedure serves its purpose and the deficits can be addressed in future work.

6.5 Summary and reflections

The issue whether the additional event and rule extensions made to the EER model increases the complexity can be addressed by splitting the analysis in distinct phases. The idea is that the earlier modeling sessions should only consider the data model. This would mean that the rule modeling sessions should not be considered before we have a reasonably stable data model. Hereafter, the analysis of rules and data could precede interchangeably. This way of working is common in object oriented techniques, i.e. initially the focus would be set on the class diagram with its attributes and associations and only thereafter would the behavioural aspects be modelled (the methods).

This work shows that CDIF can incorporate the Elmasri/Navathe EER variant which is very semantically rich and we conclude that the Data Modeling SA is of a good quality. We have used the extensibility successfully and consider it to be a powerful mechanism. All in all we believe that the standard is an interesting approach to the exchange problem between CASE tools. The experiences of working with the Meta-model and the extensibility mechanism are positive. The Integrated Meta-model of CDIF is well suited to capture rich semantic information. Overall comments address minor issues such as whether physical aspects of data modeling (e.g. AccessPath) should be mixed with conceptual information in a SA, or additions to incorporate semantic constructs from a model. It is the opinion of the author that CDIF is well suited to be a de facto standard for CASE data interchange. The future of CDIF seems promising if we consider the many new subject areas being addressed at this moment and interest from major organizations such as OMG and UML.


Bibliography


[Cha93] Chakravarthy S., Mishra D., SNOOP: An expressive Event Specification

[Cha97] Chakravarthy S., University of Florida, private conversation 97-08-11.


[Kop89] V. Kopankanas, P. Loucopoulos, Database Support for a Rule-Based Approach to Information Systems.


[EIA109] CDIF Transfer Format / Transfer Format Syntax SYNTAX.1, EIA/IS-109

[EIA110] CDIF Transfer Format / Transfer Format Encoding ENCODING.1, EIA/IS-110.

[EIA111] CDIF - Integrated Meta-model / Foundation Subject Area, EIA/IS-111.

[EIA112] CDIF - Integrated Meta-model / Common Subject Area, EIA/IS-112.


[ISO96] ISO-ANSI working draft OF SQL3, DBL:MCI-007C and X3H2-96 296C.


[RIDS97a] Rules in Database Systems, the third International Workshop on Rules in Database Systems, University of Skövde, Sweden.

[RIDS97b] Rules in Database Systems, the third International Workshop on Rules in database systems (paneldiscussion).


# Table of abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSI</td>
<td>American National Standards Institute</td>
</tr>
<tr>
<td>BR</td>
<td>Business Rule</td>
</tr>
<tr>
<td>CASE</td>
<td>Computer Aided Software Engineering</td>
</tr>
<tr>
<td>CDIF</td>
<td>Case Data Interchange Format</td>
</tr>
<tr>
<td>DBMS</td>
<td>Database Management System</td>
</tr>
<tr>
<td>DML</td>
<td>Data Manipulation Language</td>
</tr>
<tr>
<td>ECA</td>
<td>Event Condition Action</td>
</tr>
<tr>
<td>EIA</td>
<td>Electronic Institute Association</td>
</tr>
<tr>
<td>ER</td>
<td>Entity Relationship model</td>
</tr>
<tr>
<td>EER</td>
<td>Extended Entity Relationship model</td>
</tr>
<tr>
<td>IRDS</td>
<td>Information Resource Dictionary System</td>
</tr>
<tr>
<td>ISO</td>
<td>International Standards Organization</td>
</tr>
<tr>
<td>OMG</td>
<td>Object Management Group</td>
</tr>
<tr>
<td>RDBMS</td>
<td>Relational Database Management System</td>
</tr>
<tr>
<td>SA</td>
<td>Subject Area</td>
</tr>
<tr>
<td>SDLC</td>
<td>Systems Development Lifecycle</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>UoD</td>
<td>Universe of Discourse</td>
</tr>
</tbody>
</table>
Figures and tables

Figures

Figure 1: Aim and objectives...........................................................................................................4
Figure 2: Transferring business rule models ..................................................................................7
Figure 3: A synergy between business rules and data models ....................................................16
Figure 4: Rules in information systems......................................................................................21
Figure 5: The BR design process.................................................................................................23
Figure 6: The internal view of the tool.........................................................................................30
Figure 7: The CDIF subject areas...............................................................................................31
Figure 8: The foundation and common SA .................................................................................34
Figure 9: The company example ..................................................................................................52
Figure 10: The ER2 Meta-model...................................................................................................53
Figure 11: An ER2 example model...............................................................................................59
Figure 12: The Customer-Order example....................................................................................60
Figure 13: The CDIF data modeling SA ......................................................................................63
Figure 14: An example model........................................................................................................66
Figure 15: The extended Meta-model ..........................................................................................77
Figure 16: The extended Meta-model (cont.)................................................................................78

Tables

The architecture of the Meta-model.............................................................................................35
Semantic constructs in different EER variants ............................................................................49
The syntax for an event..................................................................................................................54
The syntax for a rule .....................................................................................................................56
The syntax for an action................................................................................................................57
Analysis........................................................................................................................................65
Appendix A

1. The first meta-entity that needs to be defined is the datamodel of the example:

   Meta-entity ..................... Datamodel
   Meta-attributes
      CDIFIdentifier ............. MOD01
      Name ...................... “Example”
      Brief Description ........ “An example Data Model”
      ModelType ................. “Conceptual”
      DateCreated .............. 08-06-1997
      TimeCreated .............. 01:01:59

2. After this every entity in the data model needs to be defined:

   Meta-entity ..................... Entity
   Meta-attributes
      CDIFIdentifier ............. ENT02
      Name ...................... “Employee”
      BriefDescription .......... “The employees of the company”
      EntityType ................ Kernel

   Meta-entity ..................... Entity
   Meta-attributes
      CDIFIdentifier ............. ENT03
      Name ...................... “Salaried employee”
      BriefDescription .......... “Employees with a category for compensation”
      EntityType ................ Characteristic

   Meta-entity ..................... Entity
   Meta-attributes
      CDIFIdentifier ............. ENT04
      Name ...................... “Hourly employee”
      BriefDescription .......... “Employees with a category for compensation”
      EntityType ................ Characteristic

   Meta-entity ..................... Entity
   Meta-attributes
      CDIFIdentifier ............. ENT05
      Name ...................... “Dependent”
      BriefDescription .......... “Relatives of employees”
      EntityType ................ Characteristic

   Meta-entity ..................... Entity
   Meta-attributes
      CDIFIdentifier ............. ENT06
      Name ...................... “Department”
      BriefDescription .......... “The departments where the employees work”
      EntityType ................ Kernel

   Meta-entity ..................... Entity
   Meta-attributes
      CDIFIdentifier ............. ENT07
      Name ...................... “Company”
Appendix A

BriefDescription .......... “Companies”
EntityType ................. Kernel

Meta-entity ................. Entity
Meta-attributes
   CDIFIdentifier ............ ENT08
   Name ...................... “Owner”
   BriefDescription .......... “Owner of a vehicle”
   EntityType ................. Characteristic

3. Every entity needs to be related to the data model. This is done with the Meta-relationship
   Datamodel.Collects.DatamodelObject. Note that entity is a DataModelObject:

   Meta-relationshipDatamodel.Collects.DatamodelObject
   Meta-attributes
      CDIFIdentifier ............ RELID02
      SourceEntity ............... MOD01
      Destination ................ EntityENT02

   Meta-relationship Datamodel.Collects.DatamodelObject
   Meta-attributes
      CDIFIdentifier ............ RELID03
      SourceEntity ............... MOD01
      DestinationEntity .......... ENT03

   Meta-relationship Datamodel.Collects.DatamodelObject
   Meta-attributes
      CDIFIdentifier ............ RELID04
      SourceEntity ............... MOD01
      DestinationEntity .......... ENT04

   Meta-relationship Datamodel.Collects.DatamodelObject
   Meta-attributes
      CDIFIdentifier ............ RELID05
      SourceEntity ............... MOD01
      DestinationEntity .......... ENT05

   Meta-relationship Datamodel.Collects.DatamodelObject
   Meta-attributes
      CDIFIdentifier ............ RELID06
      SourceEntity ............... MOD01
      DestinationEntity .......... ENT06

   Meta-relationship Datamodel.Collects.DatamodelObject
   Meta-attributes
      CDIFIdentifier ............ RELID07
      SourceEntity ............... MOD01
      DestinationEntity .......... ENT07

   Meta-relationship Datamodel.Collects.DatamodelObject
   Meta-attributes
      CDIFIdentifier ............ RELID08
      SourceEntity ............... MOD01
4. The entity called Employee has got two subtypes. This is defined with the Meta-entity SubtypeSet and the Meta-relationships InheritableDataModelObject.Is.SupertypeFor.SubtypeSet and InheritableDataModelObject.Is.SubtypeFor.SubtypeSet. The subtyping is defined as disjoint, which means that an employee can not both be salaried and hourly compensated. This constraint need to be stated with the optional Meta-attribute IsExclusive:

Meta-entity ................. SubtypeSet
Meta-attributes
CDIFIdentifier ............. ENT09
Name ........................ “Employee subtypes”
BriefDescription .......... “This defines the subtyping of Employees”
IsExclusive ................. TRUE

Meta-attributes
CDIFIdentifier ............. RELID09
SourceEntity ............... ENT02
DestinationEntity .......... ENT09

Meta-attributes
CDIFIdentifier ............. RELID10
SourceEntity ............... ENT03
DestinationEntity .......... ENT09

Meta-attributes
CDIFIdentifier ............. RELID11
SourceEntity ............... ENT04
DestinationEntity .......... ENT09

5. There are three relationships in the example model which needs to be defined.

Meta-entity ................. Relationship
Meta-attributes
CDIFIdentifier ............. ENT10
Name ........................ “Works_for”
BriefDescription .......... “This relates an employee to a department, i.e who works on which department”

Meta-entity ................. Relationship
Meta-attributes
CDIFIdentifier ............. ENT11
Name ........................ “Manages”
BriefDescription .......... “This relates an employee to a department i.e who manages which department”

Meta-entity ................. Relationship
Meta-attributes
CDIFIdentifier ............. ENT12
Name ........................ “Dependent of”
BriefDescription .......... “This relates an employee to his/her dependent”
6. The three relationships in the example needs to be related to the data model.

Meta-relationship...DataModel.Collects.DataModelObject
Meta-attributes
  CDIFIdentifier.............RELID12
  SourceEntity..............MOD01
  DestinationEntity........ENT10

Meta-relationship...DataModel.Collects.DataModelObject
Meta-attributes
  CDIFIdentifier.............RELID13
  SourceEntity..............MOD01
  DestinationEntity........ENT11

Meta-relationship...DataModel.Collects.DataModelObject
Meta-attributes
  CDIFIdentifier.............RELID14
  SourceEntity..............MOD01
  DestinationEntity........ENT12

7. Next, roles are defined and connected to relationships. Roleplayers are connected to their roles which in turn are connected to their entity. Note that the Elmasri/Navathe EER model only model “outer” cardinality values [EIA114, p.24]. This is solved by assigning 1:1 to the “outer” cardinalities and mapping the values of the “outer” cardinalities in the model to the “inner” cardinalities. This is the way that it should be done according to the CDIF standard body. The first relationship taken care of is “Works_for”.

Meta-entity ..................Role
Meta-attributes
  CDIFIdentifier.............ROLE01
  Name .....................“parentworksfor”
  IsMaster...................TRUE
  IsSource..................TRUE

Meta-entity ..................RolePlayer
Meta-attributes
  CDIFIdentifier.............ROLP01
  Name .....................“parentworksfor”
  MinInnerCardinality......“0”
  MaxInnerCardinality......“N”
  MinOuterCardinality......“1”
  MaxOuterCardinality......“1”

Meta-relationship.........Role.BelongsTo.Relationship
Meta-attributes
  CDIFIdentifier.............ROL_BT_REL01
  SourceEntity..............ROLE01
  DestinationEntity........ENT10

Meta-relationship.........Datamodelobject.ActsAs.RolePlayer
Meta-attributes
8. The second relationship is “Manages”:

Meta-entityRole
Meta-attributes
CDIFIdentifier ..........ROLE03
Name ...................... “parentmanages”
IsMaster .................. TRUE
IsSource .................. TRUE

Meta-entityRole
Meta-attributes
CDIFIdentifier .......... ROLP03
Name ...................... “parentmanages”
MinInnerCardinality .... “1”
MaxInnerCardinality .... “1”
MinOuterCardinality .. “1”
MaxOuterCardinality .. “1”

Meta-relationship......... Role.BelongsTo.Relationship
Meta-attributes
CDIFIdentifier .......... ROL_BT_REL03
SourceEntity .............. ROLE03
DestinationEntity ....... ENT11

Meta-relationship......... Datamodelobject.ActsAs.RolePlayer
Meta-attributes
CDIFIdentifier ............ DMO_AA_ROLP03
SourceEntity .............. ENT06
DestinationEntity ....... ROLP03

Meta-relationship......... RolePlayer.Plays.Role
Meta-attributes
CDIFIdentifier ............ ROL_P_ROLE03
SourceEntity .............. ROLP03
DestinationEntity ....... ROLE03

Meta-entityRole
Meta-attributes
CDIFIdentifier .......... ROLE04
Name ...................... “childmanages”
isMaster .................. TRUE
IsSource .................. TRUE

Meta-entity ...................... RolePlayer
Meta-attributes
CDIFIdentifier .......... OLP04
Name ...................... “childmanages”
MinInnerCardinality .... “0”
MaxInnerCardinality .... “1”
MinOuterCardinality ... “1”
MaxOuterCardinality .. “1”

Meta-relationship......... Role.BelongsTo.Relationship
Meta-attributes
CDIFIdentifier .......... ROL_BT_REL04
SourceEntity .............. ROLE04
DestinationEntity ....... ENT11

Meta-relationship......... Datamodelobject.ActsAs.RolePlayer
Meta-attributes
CDIFIdentifier ............ DMO_AA_ROLP03
SourceEntity .............. ENT02
DestinationEntity ....... ROLP04
9. The third relationship is “DependentOf”:

Meta-entity ..................... Role
Meta-attributes
CDIFIdentifier ............. ROLE05
Name ......................... “parentdependentof”
IsMaster ....................... TRUE
IsSource ....................... TRUE

Meta-entity ..................... RolePlayer
Meta-attributes
CDIFIdentifier ............. ROLP05
Name ......................... “parentdependentof”
MinInnerCardinality ... ......................... “0”
MaxInnerCardinality ...... ......................... “N”
MinOuterCardinality ... ......................... “1”
MaxOuterCardinality .. ......................... “1”

Meta-relationship ............ Role.BelongsTo.Relationship
Meta-attributes
CDIFIdentifier ............. ROL_BT_REL05
SourceEntity ................. ROLE05
DestinationEntity .......... ENT12

Meta-relationship ............ Datamodelobject.ActsAs.RolePlayer
Meta-attributes
CDIFIdentifier ............. DMO_AA_ROLP05
SourceEntity ................. ENT02
DestinationEntity .......... ROLP05

Meta-relationship ............ RolePlayer.Plays.Role
Meta-attributes
CDIFIdentifier ............. ROL_P_ROLE05
SourceEntity ................. ROLP05
DestinationEntity .......... ROLE05

Meta-entityRole
Meta-attributes
CDIFIdentifier ............. ROLE06
Name ......................... “childdependentof”
IsMaster ....................... TRUE
IsSource ....................... TRUE

Meta-entity ..................... RolePlayer
Meta-attributes
CDIFIdentifier ............. ROLP06
Name ......................... “childdependentof”
MinInnerCardinality ... ......................... “1”
10. The next step is to define all attributes. We start of with the attributes of the entity “Employee”:

Meta-entity ..................... Attribute
Meta-attributes
CDIFIdentifier .......... AT101
Name ......................... “SSn”
BriefDescription .......... “The identifying social security number”

Meta-entity ..................... Attribute
Meta-attributes
CDIFIdentifier .......... AT102
Name ......................... “Name”
BriefDescription .......... “The name of the employee”

Meta-entity ..................... Attribute
Meta-attributes
CDIFIdentifier .......... AT103
Name ......................... “Fname”
BriefDescription .......... “The first name of the employee”

Meta-entity ..................... Attribute
Meta-attributes
CDIFIdentifier .......... AT104
Name ......................... “Lname”
BriefDescription .......... “The last name of the employee”

11. These attributes needs to be related to their entity:

Meta-relationship............ DefinitionObject.Contains.ComponentObject
Meta-attributes
CDIFIdentifier .......... DO_C_CO01
SourceEntity..........ENT02
DestinationEntity......ATT01

Meta-relationship......DefinitionObject.Contains.ComponentObject
Meta-attributes
  CDIFIdentifier.........DO_C_CO02
  SourceEntity..........ENT02
  DestinationEntity.....ATT02

Meta-relationship......DefinitionObject.Contains.ComponentObject
Meta-attributes
  CDIFIdentifier.........DO_C_CO03
  SourceEntity..........ENT02
  DestinationEntity.....ATT03

Meta-relationship......DefinitionObject.Contains.ComponentObject
Meta-attributes
  CDIFIdentifier.........DO_C_CO04
  SourceEntity..........ENT02
  DestinationEntity.....ATT04

12. The attributes for the entity “Salaried employee” is defined and related:

Meta-entity .................Attribute
Meta-attributes
  CDIFIdentifier..........ATT05
  Name ....................“SSn”
  BriefDescription ......“The identifying social security number”

Meta-entity .................Attribute
Meta-attributes
  CDIFIdentifier..........ATT06
  Name ....................“Name”
  BriefDescription ......“The name of the employee”

Meta-entity .................Attribute
Meta-attributes
  CDIFIdentifier..........ATT07
  Name ....................“Fname”
  BriefDescription ......“The first name of the employee”

Meta-entity .................Attribute
Meta-attributes
  CDIFIdentifier..........ATT08
  Name ....................“Lname”
  BriefDescription ......“The last name of the employee”

Meta-entity .................Attribute
Meta-attributes
  CDIFIdentifier..........ATT14
  Name ....................“Salary”
  BriefDescription ......“Type of compensation”

Meta-relationship......DefinitionObject.Contains.ComponentObject
Appendix A

Meta-attributes
CDIFIdentifier ..........DO_C_CO05
SourceEntity ..........ENT03
DestinationEntity .......ATT05

Meta-relationship .......DefinitionObject.Contains.ComponentObject
Meta-attributes
CDIFIdentifier ..........DO_C_CO06
SourceEntity ..........ENT03
DestinationEntity .......ATT06

Meta-relationship .......DefinitionObject.Contains.ComponentObject
Meta-attributes
CDIFIdentifier ..........DO_C_CO07
SourceEntity ..........ENT03
DestinationEntity .......ATT07

Meta-relationship .......DefinitionObject.Contains.ComponentObject
Meta-attributes
CDIFIdentifier ..........DO_C_CO08
SourceEntity ..........ENT03
DestinationEntity .......ATT08

Meta-relationship .......DefinitionObject.Contains.ComponentObject
Meta-attributes
CDIFIdentifier ..........DO_C_CO14
SourceEntity ..........ENT03
DestinationEntity .......ATT14

13. The attributes for the entity “Salaried employee” needs to be defined as “inherited” from employee:
Meta-relationship .........Attribute.IsInheritedFrom.Attribute
Meta-attributes
CDIFIdentifier ..........A_IIF_A01
SourceEntity ..........ATT05
DestinationEntity .......ATT01

Meta-relationshipAttribute.IsInheritedFrom.Attribute
Meta-attributes
CDIFIdentifier ..........A_IIF_A02
SourceEntity ..........ATT06
DestinationEntity .......ATT02

Meta-relationshipAttribute.IsInheritedFrom.Attribute
Meta-attributes
CDIFIdentifier ..........A_IIF_A03
SourceEntity ..........ATT07
DestinationEntity .......ATT03

Meta-relationshipAttribute.IsInheritedFrom.Attribute
Meta-attributes
CDIFIdentifier ..........A_IIF_A04
SourceEntity ..........ATT08
DestinationEntity .......ATT04
14. The attributes for the entity “Hourly employee” is defined and related, including the local attribute “PayScale”.

Meta-entity ..................... Attribute
Meta-attributes
  CDIFIdentifier .......... ATT09
  Name .................... “SSn”
  BriefDescription ........ “The identifying social security number”

Meta-entity ..................... Attribute
Meta-attributes
  CDIFIdentifier .......... ATT010
  Name .................... “Name”
  BriefDescription ........ “The name of the employee”

Meta-entity ..................... Attribute
Meta-attributes
  CDIFIdentifier .......... ATT011
  Name .................... “Fname”
  BriefDescription ........ “The first name of the employee”

Meta-entity ..................... Attribute
Meta-attributes
  CDIFIdentifier .......... ATT012
  Name .................... “Lname”
  BriefDescription ........ “The last name of the employee”

Meta-entity ..................... Attribute
Meta-attributes
  CDIFIdentifier .......... ATT013
  Name .................... “PayScale”
  BriefDescription ........ “Type of compensation”

Meta-relationship............ DefinitionObject.Contains.ComponentObject
Meta-attributes
  CDIFIdentifier .......... DO_C_CO06
  SourceEntity .......... ENT04
  DestinationEntity .... ATT09

Meta-relationship............ DefinitionObject.Contains.ComponentObject
Meta-attributes
  CDIFIdentifier .......... DO_C_CO07
  SourceEntity .......... ENT04
  DestinationEntity .... ATT10

Meta-relationship............ DefinitionObject.Contains.ComponentObject
Meta-attributes
  CDIFIdentifier .......... DO_C_CO08
  SourceEntity .......... ENT04
  DestinationEntity .... ATT11

Meta-relationship............ DefinitionObject.Contains.ComponentObject
Meta-attributes
15. The attributes for the entity “hourly employee” needs to be defined as “inherited” from employee.

\[
\begin{align*}
\text{CDIFIdentifier} & \quad \text{DO_C_CO010} \\
\text{SourceEntity} & \quad \text{ENT04} \\
\text{DestinationEntity} & \quad \text{ATT10} \\
\end{align*}
\]

\[
\text{Meta-relationship} \quad \text{Attribute.IsInheritedFrom.Attribute} \\
\text{Meta-attributes}
\begin{align*}
\text{CDIFIdentifier} & \quad \text{A_IIF_A05} \\
\text{SourceEntity} & \quad \text{ATT09} \\
\text{DestinationEntity} & \quad \text{ATT01} \\
\end{align*}
\]

\[
\text{Meta-relationship} \quad \text{Attribute.IsInheritedFrom.Attribute} \\
\text{Meta-attributes}
\begin{align*}
\text{CDIFIdentifier} & \quad \text{A_IIF_A06} \\
\text{SourceEntity} & \quad \text{ATT10} \\
\text{DestinationEntity} & \quad \text{ATT02} \\
\end{align*}
\]

\[
\text{Meta-relationship} \quad \text{Attribute.IsInheritedFrom.Attribute} \\
\text{Meta-attributes}
\begin{align*}
\text{CDIFIdentifier} & \quad \text{A_IIF_A07} \\
\text{SourceEntity} & \quad \text{ATT11} \\
\text{DestinationEntity} & \quad \text{ATT03} \\
\end{align*}
\]

\[
\text{Meta-relationship} \quad \text{Attribute.IsInheritedFrom.Attribute} \\
\text{Meta-attributes}
\begin{align*}
\text{CDIFIdentifier} & \quad \text{A_IIF_A08} \\
\text{SourceEntity} & \quad \text{ATT12} \\
\text{DestinationEntity} & \quad \text{ATT04} \\
\end{align*}
\]

16. The attribute of the relationship “Manages” needs to be defined and related:

\[
\begin{align*}
\text{Meta-entity} & \quad \text{Attribute} \\
\text{Meta-attributes}
\end{align*}
\]

\[
\begin{align*}
\text{CDIFIdentifier} & \quad \text{ATT15} \\
\text{Name} & \quad \text{Startdate} \\
\text{BriefDescription} & \quad \text{Startdate} \\
\end{align*}
\]

\[
\text{Meta-relationship} \quad \text{DefinitionObject.Contains.ComponentObject} \\
\text{Meta-attributes}
\begin{align*}
\text{CDIFIdentifier} & \quad \text{DO_C_CO06} \\
\text{SourceEntity} & \quad \text{ENT11} \\
\text{DestinationEntity} & \quad \text{ATT15} \\
\end{align*}
\]

17. The attributes of the entity “Department” needs to be defined:
Appendix A

18. The attribute “Number_of” needs to be defined as derived:

Meta-entity ..................... ProjectedAttribute
Meta-attributes
CDIFIdentifier ............. ATT18
Name .......................... “Nr_of_emp”
BriefDescription ........... “The total number of employees on a department”

Meta-relationship........... ProjectedAttribute.IsProjectionOf.Attribute
Meta-attributes
CDIFIdentifier ............. PA_IPO_A01
SourceEntity ............... ATT18
DestinationEntity.......... ATT16

19. The attributes of the entity “Dependent” needs to be defined:

Meta-entity ..................... Attribute
Meta-attributes
CDIFIdentifier ............. ATT19
Name .......................... “Name”
BriefDescription ........... “The Name of a dependent”

Meta-relationship........... DefinitionObject.Contains.ComponentObject
Meta-attributes
CDIFIdentifier ............. DO_C_CO10
SourceEntity ............... ENT05
DestinationEntity......... ATT19

Meta-entity ..................... Attribute
Meta-attributes
Appendix A

CDIFIdentifier ..........ATT20
Name ......................“Relationship”
BriefDescription ..........“The type of Relationship”

Meta-relationship ..........DefinitionObject.Contains.ComponentObject
Meta-attributes
CDIFIdentifier ..........DO_C_CO11
SourceEntity ..............ENT05
DestinationEntity .......ATT20

20. Next, we need to define the unique keys for the entities in the model. This is done with the Meta-entity Candidatekey and the Meta-relationships Entity.IsIdentified_by.Key and Key.Incorporates.Attribute.

Meta-entity ..............Candidatekey
Meta-attributes
CDIFIdentifier ..........CKEY01
Name ......................“Ssn”
BriefDescription ..........“The unique key of Employee”

Meta-relationship ..........Entity.IsIdentified_by.Key
Meta-attributes
CDIFIdentifier ..........E_II_K01
SourceEntity ..............ENT02
DestinationEntity .......CKEY01

Meta-relationship ..........Key.Incorporates.Attribute
Meta-attributes
CDIFIdentifier ..........K_I_A01
SourceEntity ..............CKEY01
DestinationEntity .......ATT01

Meta-entity ..............Candidatekey
Meta-attributes
CDIFIdentifier ..........CKEY02
Name ......................“Number”
BriefDescription ..........“The unique key of Department”

Meta-relationship ..........Entity.IsIdentified_by.Key
Meta-attributes
CDIFIdentifier ..........E_II_K02
SourceEntity ..............ENT06
DestinationEntity .......CKEY02

Meta-relationship ..........Key.Incorporates.Attribute
Meta-attributes
CDIFIdentifier ..........K_I_A02
SourceEntity ..............CKEY02
DestinationEntity .......ATT16

Meta-entity ..............Candidatekey
Meta-attributes
CDIFIdentifier ..........CKEY02
Name ......................“Number”
Appendix A

BriefDescription ............. “The unique key of Department”

Meta-relationship............ Entity.IsIdentified_by.Key
Meta-attributes
CDIFIdentifier ............... E_IT_K02
SourceEntity ................. ENT06
DestinationEntity .......... CKEY02

Meta-relationship............ Key.Incorporates.Attribute
Meta-attributes
  CDIFIdentifier .............. K_I_A02
  SourceEntity ............... CKEY02
  DestinationEntity ...... ATT16

Meta-entity ..................... Candidatekey
Meta-attributes
  CDIFIdentifier ............. CKEY03
  Name ...................... “Name”
  BriefDescription .......... “The unique key of Dependent”

Meta-relationship............ Entity.IsIdentified_by.Key
Meta-attributes
  CDIFIdentifier .............. E_IT_K03
  SourceEntity ............... ENT05
  DestinationEntity ...... CKEY03

Meta-relationship............ Key.Incorporates.Attribute
Meta-attributes
  CDIFIdentifier .............. K_I_A03
  SourceEntity ............... CKEY03
  DestinationEntity ...... ATT19