





**run Software-Werkstatt GmbH**  
**Köpenicker Strasse 325**  
**12555 Berlin**

[www.run-software.com](http://www.run-software.com)

Tel: +49 (30) 65762791  
e-mail: [run@run-software.com](mailto:run@run-software.com)

Berlin, October 2011

# Content

<b>1</b>	<b>Terminology Model II .....</b>	<b>5</b>
<b>2</b>	<b>Knowledge Presentation .....</b>	<b>7</b>
2.1	Objects .....	8
2.1.1	Individual and general objects.....	9
2.1.2	Intensional and extensional aspect.....	10
2.2	Object classes - grouping objects.....	11
2.3	Object types - what objects are .....	12
2.4	Object collections.....	14
2.5	Classification - creating class hierarchies.....	15
2.5.1	Classification aspects.....	17
2.6	Rules - how objects behave.....	19
2.7	Causalities - when objects behave .....	20
<b>3</b>	<b>Terminology Model Object Types.....</b>	<b>21</b>
3.1	Concepts and terms.....	22
3.2	Feature categories .....	23
3.2.1	Rules .....	25
3.2.2	Events .....	26
3.2.3	Causality.....	27
3.3	Item .....	28
3.3.1	Type .....	30
3.3.2	Property.....	39
<b>4</b>	<b>Terminology Model Reference.....</b>	<b>46</b>
4.1	Terminology model rules.....	47
4.2	From terminology model to database .....	49
4.3	Terminology Model Object Types .....	50
4.3.1	AggregationType - Aggregation type .....	53
4.3.2	Association - Associations .....	54
4.3.3	Category - Category.....	55
4.3.4	Causality - Causality .....	57
4.3.5	Characteristic - Characteristic.....	58
4.3.6	Classification - Classifications.....	59
4.3.7	Concept - Concepts and terms .....	61
4.3.8	DocumentReference - Document reference .....	64
4.3.9	Event - Events.....	66
4.3.10	Feature - Feature categories.....	68
4.3.11	FixedCharacteristic - Fixed characteristic .....	70

4.3.12	Generalization - Generalization.....	71
4.3.13	Item - Item .....	72
4.3.14	Key - Keys .....	76
4.3.15	ObjectType - Object types.....	77
4.3.16	Part - Parts .....	82
4.3.17	Property - Property.....	83
4.3.18	Rule - Rules .....	87
4.3.19	TerminologyModel - Terminology model.....	89
4.3.20	TerminologyModule - Terminology Module.....	91
4.3.21	Type - Type .....	94
4.4	Classifications .....	95
4.4.1	ElementaryTypes - Elementary types .....	96
4.4.2	EventTypes - Event types .....	98
4.4.3	FeatureTypes - Feature types.....	101
4.4.4	ImportanceLevels - Importance levels .....	104
4.4.5	RuleTypes - Rule types.....	105
<b>5</b>	<b>References .....</b>	<b>106</b>

# 1 Terminology Model II

Bringing human language and data modeling closer together is an important step in modeling real world phenomena in IT applications. For this purpose, the terminology model provides a conceptual and structured terminology definition.

Defining a terminology model is based on principles of expressing and structuring human knowledge. The principles have been derived from the way knowledge is expressed in human language, since this has been turned out to be the best practice for hundredth of years.

Concept and feature are the key terms for defining terminology models, where a number of concepts and its features are defined. In order to provide detailed terminology definitions, concepts and features are assigned to several categories (concept or feature categories).

The "Terminology Model II" is the continuation of work that has been represented in "[Terminology Model I](#)" [TM]. In order to provide more precise definitions, several extensions have been made in the model structure.

The paper consists of three parts. The first part introduces the main concepts the terminology model deals with (problem analysis). The second part provides complete definitions and explanations for all concepts referred to in this paper (structuring concepts) and the third part provides a detailed definition of these concepts as structured terminology model (defining details). Thus, the paper not only defines the terminology model as such, but is an example also for developing a terminology model starting with high level problem analysis, i.e. the paper is doing exactly what it is describing.

In order to bring together terminology and data modeling, terminology standards as described in ISO 1087 and ISO 704 [1087] are referred to as terminological source, where **concepts** and **concept relations** have been defined in order to build **concept systems**. The ODMG Object Database Standard 2003 [ODMG] is used as base for object database modeling, since this approach is more advanced as the Entity Relationship Model. [Unified Database Theory](#) [UDT] is referred to as detailed definition of **data items**, **instances** and **collections** and representation of data in different levels of database models.

Further on, the acronym **TM** will be used when referring to Terminology Model II as model or as paper.

### **What is a terminology model**

A terminology model is a mean for recording and presenting expert knowledge of a certain area. TM provides both, a method for discovering subject expert knowledge and a structured and formalized presentation of expert knowledge.

TM is a method for describing expert knowledge within a certain subject field, which covers two aspects:

- Terminology aspect - defining used concepts (term, meaning, usage etc.) in a subject field
- Model aspect - provides a formal structured presentation of concepts, which fulfills basic requirements of a database schema

The goal of TM on the one hand is, to provide detailed definitions and, on the other hand, to support all phases of an IT application development process. Thus, the TM also acts as interface between subject area experts and IT technicians.

Since the goal of TM is providing structured knowledge representation, the paper investigates knowledge concepts and principles for expressing knowledge by means of human language.

The paper will finally end up in a rather complex structure for TM, but many relevant knowledge aspects are covered by a few TM concepts. Hence, different TM levels will be defined which can be used for different purposes or in different development stages.

## 2 Knowledge Presentation

Terminology models reflect principles of knowledge presentation by means of human language, i.e. here, knowledge presentation is considered as one way of expressing knowledge in terms of human language.

There are other ways of expressing knowledge like formulas or pictures. Transferring knowledge between human beings, however, is often based on human language consisting of terms with an agreed meaning. Some essential principals of human language are subject of TM and are described below.

Subject of terminology studies is defining the meaning of terms used in human language, the concepts. Thus, terminology is an essential part of knowledge presentation by means of human language. A common way of defining concepts and terns is described in [1087], which defines concepts and concept relations as important part of terminology definitions.

TM is based on concepts and concept relations as being defined in [1087]. But terminology is more than defining concepts. Terminology provides common rules in order to express specific views to the world. A specific view to the world, however, expresses the way experts have experienced a specific subject area.

It is not the intention of this paper to provide a complete definition of knowledge presentation. Instead, it will focus on several aspects of knowledge presentation, which are subject of a knowledge presentation by means of terminology. Several principles of expressing knowledge by means of human language are in focus of the paper:

- Discovering objects
- Building objects classes
- Defining object types as abstraction of objects
- Grouping objects in object collections
- Classifying object in order to create object collections
- Describing behavior of objects
- Describing causalities by means of cause and reaction

These aspects of knowledge presentation have been subject of the development of TM. This chapter will introduce concepts describing these aspects without defining them in detail, i.e. it only explains, what is the meaning of these concepts. A more precise definition of the concepts is given in chapter "Terminology Model Object Types".

## 2.1 Objects

An base for expressing knowledge is the concept of objects. Real world phenomena are often called **objects**, regardless whether objects exist as "things" or just as ideal concepts. In any case, objects are described by means of associated concepts.

Without objects, a TM would not exist, i.e. no objects - no TM. Practically, this means that TM is useful only, when considering objects from different perspectives. On the other hand, also human language is mainly based on objects. Considering nouns as typical references to objects, the importance of objects in human language becomes obvious.

The concept of an object includes objects of any level of detail (or complexity). Thus, also an object collection, a set of objects could be considered as object. Similar, details of an object, its properties, can be considered as objects, again.

## 2.1.1 Individual and general objects

Human language provides means for describing individual objects, but also general objects. An **individual object** is a concept that reflects a particular object instance (real world phenomenon or idea). Typically, individual objects are represented as list of individual properties (property values). A **general object** is a concept that defines common features of a group of individual objects. E.g. *Paul Miller* refers to an individual object, while *person* denotes a general object.

Since modeling is an abstract issue, in TM mainly general objects are considered. In some cases, terminology, however, is defined in the context of an individual object, e.g. an enterprise, i.e. concepts and terms get its specific meaning in the context of this enterprise.

Typically, the meaning of general objects is transferred by means of examples ("This is a tree"). After being introduced to a sufficient number of *trees*, even a child is able to recognize a *tree* without being able to define the concept of the general object *tree*. Thus, many concepts for general objects are known, but not defined in detail. In order to transfer knowledge between subject area experts and IT experts it is, however, not sufficient to "know", what a general object is, but it has to be defined precisely.

Knowledge presentation requires detailed definitions of general objects. Borderlines for object classes have to be defined in order to distinguish objects referring to different concepts.

## 2.1.2 Intensional and extensional aspect

A specific view to a group of individual objects depends on the higher context that created this view. E.g. the view to a *person* is different from a statistician's point of view or from the one of a tax authority. Thus, specific views to objects may be defined within a certain context or as subordinated concepts of another concept. Such a view to an object defines the intensional aspect of the related concept, the object type.

An **object type** is a concept that defines a specific view to a group of individual objects, i.e. it defines the intensional aspect of a general object by means of subordinated concepts with different roles.

Since there are different views possible to any group of individual objects, several object type definitions may apply to an individual object. Considering an object as an entity in the real world (even though it becomes always difficult to draw borderlines for an object), each object type definition applying to an individual object creates an image of the object called **object instance**. Since different views to objects are possible, individual objects may be associated with different object types. Thus, several types may apply to an individual object and any number of object instances may exist for an individual object (e.g. the *person Paul Miller* might be reflected as *student*, *patient* and twice as *employee*), i.e. there is a 1:M relationship between individual objects and object instances.

## 2.2 Object classes - grouping objects

Objects with common properties are often "collected" in **object classes**, which represent general object collections. Classes are associated with names (terms, designations) that relate to the class concept (intension) on the one hand and to the class extension on the other hand.

Often, classes are based on intensional definitions by describing, what type of objects a class consists of, i.e. by defining an object type. But there exist also extensional class definitions (especially for conceptual objects), where the elements of a class are listed explicitly. Hence, class definitions include two aspects

- **Extensional definition** - describes the class by referring to objects belonging to the class
- **Intensional definition** - provides a conceptual definition of the object type in terms of typical characteristics of the objects belonging to a class.

Classes are not given by nature, but are an expression of human knowledge and the specific view to a certain subject area. Thus, an object may belong to any number of classes, which just express different views to the objects. Usually, classes are referred to as nouns that are used as terms designated to particular class concepts.

TM will not consider classes as such, but rather these two class aspects represented by object types (intensional) and object collections (extensional).

## 2.3 Object types - what objects are

Intensional class definitions may be defined as objects types. Typing reflects a knowledge aspect based on class definitions. In order to describe details of an individual object, language refers to object properties (*first name is Paul, second name is Miller, weight is 75 kilograms* or *he has got children Jane, Anton and Mary*). In order to refer to object instance data, property names and its meaning have to be defined (*first name, second name, weight* or *children*). Thus, an object type is typically defined by a number of property definitions.

**Object types** are concepts that are defined by a number of fixed and variable characteristics, which describe relevant properties for objects of a given class and concept relations that describe type of relations to other objects. Characteristics and concept relations defined for an object type are called **properties**. Thus, object types reflect a specific perspective to a set of objects associated with this type and provide an intensional definition for objects in the object class associated with the object type.

Object properties may be divided into four categories:

- Generalization - describes a concept that relates to a more general object type.
- Characteristic - describes a fixed or variable characteristic of an object
- Part (of) - describes the parts an object consists of
- Association - describes the relations to other objects

Those property categories are defined in concept systems as concept relations and characteristics [1087]. In data modeling they are known as inheritance, attributes and relationships [ODMG]. [UML] refers to property categories as generalization, attributes, aggregation and association. More relations are known between concepts as *near or related* concepts, i.e. this list of property categories is not complete. But the categories mentioned above are those that are considered as most relevant property categories for defining a terminology model as bridge between subject area and IT experts.

There is not always a clear distinction between property categories defined for an object type. E.g. whether a wheel is a part of a car or whether it is associated with a car depends on the view one has to the car. Thus, an object type does not define an object as such, but a certain view to objects of a given class. Moreover, properties do not describe all the properties of an object, but properties relevant from a certain perspective, only - the expert's view.

Many properties refer to other objects, which, again, are associated with an object type that defines properties etc. Thus, there is an alternative *type - property - type - property ...* relation, which provides a method of defining a terminology model in a consistent way until any level of detail. The alternating sequence usually ends,

when a property refers to a type that does not have properties (elementary type or classification).

## 2.4 Object collections

An object collection refers to set of objects. There might be different reasons for collecting objects in an object collection. Those reasons mainly define the concept of an object collection. Thus, the way, objects are assigned to object collections, is a relevant aspect of expressing human knowledge.

Objects in object collections may require different treatment, as *paid* and *unpaid invoices* are treated differently. The way, an object collections or objects within the collection behave is also a relevant knowledge aspect.

An **object collection** is a concept that defines a set of individual objects. Object collections can be defined as individual object collections (*unpaid invoices of run software*) or as general object collections (*unpaid invoices*). An **individual object collection** is a concept that describes a specific set of individual objects. A **general object collection** is a concept that defines those individual objects, which may become element of the object collection, i.e. the general object collection describes the extensional aspect of an object collection. General object collections could also be defined by intensional definition, i.e. by describing the role of the collection (*children of a person*).

Usually, individual objects refer to individual object collections, i.e. a specific enterprise has got a specific collection of *unpaid invoices* according to the definition of a general object collection *unpaid invoices*. Thus, an **individual object collection** refers to a specific set of individual objects, which represent the value or state of the object collection [UDT].

In general, human language does not refer to object collections, but to object instance collections, i.e. not objects are collected, but object instances. Thus, a *person* being employed in two *companies* appears twice as *employee*, i.e. it results in two *employee* object instances. Counting *employees* in this case will result in a number greater than the number of existing individual objects (*persons*) in the same context.

When all objects in an object collection belong to the same object type (e.g. *person*), the collection becomes a **typed object collection**. Classes are specific typed object collections, where the object collection consists of all individual objects of a given object type.

Considering object collections also relates to aggregation process in IT. A more sophisticated way of defining object collections will be discussed "Classifications - creating class hierarchies".

## 2.5 Classification - creating class hierarchies

In order to provide aggregated information about objects belonging to different categories, individual objects can be classified by means of classifications. A **classification** is a concept, which provides a mean for dividing an object collection into distinct subsets. A classification consists of a set of categories, which create subsets (classified collections) for the object collection by associating each individual object of the object collection with exactly one category.

In general, it is also possible to classify individual objects by other individual objects (ad-hoc classifications). It depends on the specific view and the subject area, when to use a classification and when to use object references.

### Categories

**Categories** describe the rules for dividing an object collection into subsets. Usually, categories are defined as **general categories** not referring to a certain set of objects. In principle, it would also be possible to define individual categories that refer to a set of individual objects. This happens, when defining (hierarchical) classifications, which explicitly refer to related objects. E.g. a continent classification may list the countries belonging to each continent for each category in the classification.

There were many discussions whether categories are metadata (i.e. defined on the model level) or data stored in the database. Considering geographical classifications, a category refers to a certain country or geographical area, which, indeed, is an individual object and hence, defined on the data level. On the other hand, many statistics do not care about the individual object aspect of geographical objects but use those for classifying companies or persons. Thus, for a statistician, geographical areas appear rather as categories than as individual objects. Finally, both is right. It just depends on the way of reflecting reality.

### Typed and untyped classification

Often, classifications may apply to individual objects of a certain object type, only. Applying a *color* classification referring to *colors* as categories to an object collection requires that individual objects in the collection have a *color*, i.e. the classification may apply on "*colored objects*" (object type), only. Classifications applying to individual objects of a given object type are called **typed classifications**.

A typed classification produces a number of subsets (but not necessarily subclasses), when applying to an individual object collection. Since individual objects in the object collection belong to the same object type, the subset created for each category can be defined as condition, which individual objects have to fulfill for belonging to a given category (e.g. all *person* objects with an age between 20 and 29 belong to the *twenties* category).

Practically, classifications also may apply to object collections containing individual objects of any object type (e.g. by assigning individual object explicitly to categories). Classification not requiring individual object of a specific object type are called **untyped classifications**. Usually, classifications refer to properties that all individual objects must have in order to be classified, which means, that object to be classified usually have a common object type that becomes the object type of the classification.

### **Typed category**

Categories of typed classifications may create subclasses. Considering a *person* classification consisting of categories *male* and *female*, may create not only person subsets, but also specialized person classes as *men* and *women*. In order to produce specialized subclasses, the category requires an object type, which has to inherit from the classifications object type. Categories producing subclasses rather than subsets are called **typed categories**.

### **Hierarchical classifications**

Since a category creates a new object collection, another (sub) classification may apply to this subset in order to divide it again. Thus, by combining classifications, **hierarchical classification** can be defined. A set of hierarchical classifications may form a **classification system**, where all (hierarchical) classifications included refer to a common set of (flat) classifications.

Within a hierarchical classification it becomes obvious, the each category (except the ones on lowest level) is a classification, again. Hence, categories are considered in general as classifications.

### **Classification properties**

Some classifications require additional characteristics in order to describe categories properly, e.g. the *duration of study* for *education* classifications. Thus, classifications support defining a number of properties for their categories.

Within a hierarchical classification, extension properties are supposed to exist for all categories referred to in the hierarchical classification or on all categories defined on the lowest hierarchy level.

## 2.5.1 Classification aspects

Classifications are used in different ways, which express different aspects of a classification. Depending on the specific view, the importance of classification aspects for different classification may differ.

### General classification

A general or conceptual classification consists of a list of (hierarchical) categories, which express a specific interest or view of the individual defining the classification categories. Defining proper classifications is an important issue, which sometimes may take years.

Typically, classifications apply on objects of a given type, i.e. general classifications are considered as typed classifications.

### Classifying individual objects

One important purpose for defining classifications is classifying individual objects, i.e. assigning objects to different well defined categories in order to determine the behavior of objects more precisely (only *persons* with *sex female* are able to get *children*). Thus, classifying individual objects and creating specialized object types are similar processes.

### Creating specialized object types

Indeed, by classifying objects, these objects become more specific, i.e. they are associated with a more specific object type. Practically, categories are often not considered as sub types, but simply as categories (*male*, *female*) assigned to objects of a given object type (*person*). Considering, however, biological classifications, it becomes obvious, that each category defines a more specialized object type (animals --> birds --> swans).

### Classified collections

Another common purpose of classifications is to divide individual object collections into a set of (hierarchical) subsets, i.e. a classification provides a collection schema for creating a number of **classified collections** from an individual object collection.

### Aggregation

Classified collections are often used for getting derived (aggregated) information from those collections. In order to define properties for aggregated values (*sum of income*, *count*), classified collections are considered as individual objects belonging to the same object type, which defines the characteristics for classified collections. Since different views to classified collections are possible, any number of

object types might be defined for a set of classified collection, i.e. for a classification, which are called **aggregation types**.

## 2.6 Rules - how objects behave

Object types and properties support a static definition of objects and object classes. An advanced knowledge approach, however, is to describe, how objects or properties behave. Typically, the behavior of objects or properties is described as common behavior of objects of a given object type ("*birds* are able to *fly*" or "*things* are able to *fall*") or as common behavior of a property defined for an object type (*salary* must not exceed \$ 5000).

A **rule** is a concept, that describes the common behavior for individual objects of a given object type or for individual properties. In principle, rules could also be described as individual concepts, i.e. as the specific behavior of an individual object or property. Practically, this happens, when defining a terminology model for a specific enterprise. Individual rules (e.g. reflected in an application for this enterprise) are, however, not described for individual instances, since the behavior of the enterprise is an internal behavior. Considering enterprises in a broader context, general rules for enterprise object might be defined. Hence, TM considers rules as general concepts, only.

Rules are used for different purposes, e.g. for defining constraints or operations for deriving information from other sources.

## 2.7 Causalities - when objects behave

Describing causalities (or reactions) is an even more advanced approach. Causalities describe the cause or reason, which activates a certain behavior (action). Describing causalities reduces the human interaction in a system or application, because the application can detect automatically, what is necessary to do and when.

The methods of describing causalities are not yet developed very well. Three ways of describing causalities are considered in TM:

- Describing causalities as state or state transitions that cause an action (activating a rule). E.g. when a person has been born (cause) it must be registered in the birth register (action).
- Describing causalities as consequence of history e.g. sending a mail (action) when a person has birthday. In fact, this does not differ in principle from the first topic, but time plays a special role and thus, it becomes useful to consider temporal causalities separately.
- Often, causalities are reflected in terms of process events as an individual object has been updated, created or deleted. Usually, process events cannot be expressed in terms of object state transitions and are, typically, considered as internal or system events.

There are not so many areas dealing with causalities compared to static concept definitions. On the other hand the introduction of process events in database systems and GUI environments has already shown powerful features based on event handling.

## 3 Terminology Model Object Types

In order to define a terminology model, several concepts have to be defined, which can be assigned to different concept categories. The following topics will introduce different concept categories by explaining the meaning of these concept categories. In this chapter, precise definitions of concept categories are provided. Details of terminology model object types are described in "Terminology Model Reference".

In the previous chapter, the following relevant concept categories have been discovered:

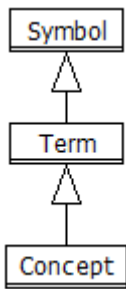
- Object
- Class
- Object collection
- Object type
- Classification
- Category
- Aggregation type
- Behavior (rules)
- Causalities

Although objects are subject of terminology models, they will not directly play a role within the terminology model. Not the objects itself, but different ways of representing knowledge about objects are in focus of TM. In general, TM distinguishes between concepts relating to objects or part of it (items) and the features, an item concept provides.

TM defines a hierarchy of items and its features. Hence, this chapter mainly introduces categories for items and features.

### 3.1 Concepts and terms

An essential point of transferring knowledge is common understanding and referring to proper and well-defined terms.



Transferring knowledge, i.e. communication, is based on sequences of symbols. A **symbol** is any kind of signal (visual, acoustical etc.) with an agreed underlying meaning passed between sender(s) and receiver(s). A **term** (single word or sequence of words) is a symbol based on a sequence of letters and numbers. The meaning of a term is not necessarily unique but it becomes unique in a certain context. TM considers terms only as mean of expressing knowledge. Other symbols as pictures or sounds are not considered in TM.

Different terms may have the same meaning (**synonyms**). Synonyms might be defined when being used within a subject area, but also, when terminology changes. In the latter case synonyms allow mapping old terms to newer terms.

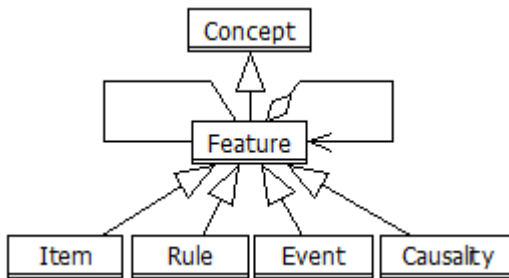
Terms are used for referring to concepts. [1087] defines a concept as "unit of knowledge created by a unique combination of characteristics". TM needs to generalize this definition slightly. Here, a **concept** is a unit of knowledge which is referred to by a term and which is defined by textual description. Thus, a concept is a term with a meaning that reflects a part of reality or an idea, but concepts do not necessarily refer to characteristics.

Simply said, any term that has got a definition or explanation is considered as concept in TM. There are no formal criteria to measure the quality of the definition or explanation. Thus, it depends on the TM developer, how good a terminology model finally is.

As mentioned in the previous chapters, concepts applying to objects may appear as individual or general concepts. TM mainly considers general concepts, although the terminology model itself may refer to an individual object (e.g. when defining the terminology model for a specific enterprise).

## 3.2 Feature categories

Many concepts refer to subordinated concepts (e.g. via concept relations). Subordinated concepts that belong to exactly one concept (parent or owner) are called **feature**. Features may appear as directly subordinated concepts (e.g. characteristics) or as concept relation referring to other concepts.



When a feature refers to another concept, the feature still belongs to its owner, but not the referenced concept. E.g. a constraint hat refers to a rule belongs to its owning concept, but not the rule referenced. Features, that define concept relations, are also called feature relations. A **feature relation** is a concept

relation that refers to a feature defined as feature of another concept. Usually, feature relations do not contain subordinated features, but the relation to exactly one other feature. There are, however, also features, that define a feature relation by referring to the feature of another concept and providing own features. Thus, TM does not strictly distinguish between feature and feature relation, but refers to feature and feature relation in order to refer to the owner or relation aspect of a feature.

The feature relation is a concept describing the use of a defined feature within another feature, while subordinated features are part of its parent feature. E.g. the validation rule (constraint) for the *salary* property of an *employee* (feature relation) refers to a rule (feature) defined for object type *employee* (checks, whether an *employee* does not earn more than his boss). The rule is defined as feature of the object type *employee* but it is referenced from the property *salary* as constraint (feature relation). Thus, the feature relation defines the role the referenced feature plays in a specific environment.

Defining the details of a concept by means of features allows expressing detailed knowledge not only about a subject area, but also about object types, properties and object collections.

Features may get subordinated features, again. Thus, features may form a hierarchy. The meaning of the term referring to a feature is uniquely defined within the owning feature, only.

Since features define the meaning of a term in the context of another feature, the same term might be assigned to features owned by another feature (e.g. properties assigned to different object types may use the same name with different

meaning, as *size* property for *person* and *company*). Thus, the meaning of a feature, i.e. its concept definition, may differ depending on the owning feature.

In a terminology model, each concept is defined at least in the context of the individual or general object, which provides the frame for the terminology model. Hence, instead of defining concepts, TM defines features and feature hierarchies.

In order to simplify the terminology model, features levels are assigned to features. Level 1 features allow defining a simple terminology model and include object types, classifications and properties. Level 2 features describe an enhanced terminology model that also includes rules, causalities and keys.

In [1087], features are not defined as such. There, object relations and characteristics are defined as details of a concept, but those are not considered as subordinated concepts. In [ODMG], properties and behavior (what objects of a class are able to do) are considered as features for object types, as well as extents defining object collections or keys. In addition, TM supports a number of feature categories, which play an important role when expressing knowledge.

- Rule
- Event
- Causality
- Item

Items summarize a number of more specific features. Rule, event and causality are features defined for items as well as constraints.

Exceptions are defined as features in [ODMG], too, in order to describe, how objects of a given type behave in unexpected situations. Since exceptions seem to be rather an implementation issue than a conceptual one, exceptions are not considered as feature categories in TM.

### 3.2.1 Rules

Object types and properties support a static definition of objects and object classes. An advanced knowledge approach, however, is to describe, how objects behave. Typically, the behavior of objects is described as common behavior of objects of a given object type ("*birds* are able to *fly*" or "*things* are able to *fall*").

One typical approach is describing behavior as rules. A **rule** is a feature that describes how object or property instances change from one state to the other or how objects interact with other objects. Rules are referred to for different purposes (rule categories):

- **Constraints** are used as validation rules for objects and properties
- State **transitions** describe the way how objects change or interact
- **Operations** provide derived information

Defining rules is an advanced approach and not typically used when starting defining concepts. Later on, it becomes, however, important, because building applications is impossible without knowing the rules, according to which objects in the application behave and interact with each other.

Typically, rules apply on single values, object instances or object collections. In order to control rules, any number of parameters (properties) might be passed to a rule. Depending on the rule category, it may change the state of the instance it applies on or return a result property. Parameters and result are features of the rule.

There is no clear distinction between rule and characteristic or property, since characteristics (like *age*) might be also expressed as operation rule, which returns the *age*. Again, it depends on the experts view, whether *age* is considered more as characteristic of person or as rule evaluating the *age* or as both.

## 3.2.2 Events

An **event** describes a cause or reason, which may activate an action (rule). There is a difference between individual and general events. An **individual event** is something that had really happened, while a general event is an event that may happen. TM mainly considers general events.

Usually, general events are referenced from within causalities. The same general event might be referred to from within different causalities, in which case all related actions are called, when an individual event happens.

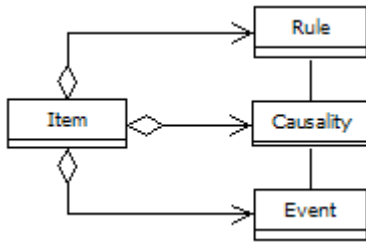
**General events** are features describing relevant state transitions. Depending on different categories of state transitions, three categories of events may be defined:

- **Instance events** - are defined as relevant instance state transitions that may cause a reaction (activating a rule), e.g. when a person has married (cause) this has to be registered (action).
- **Time events** - are defined as relevant time state transitions (e.g. a person's birthday) that require appropriate actions (as sending a mail). This does not differ in principle from object state transitions, but time plays a special role and thus, it becomes useful to consider time-dependent reactions separately.
- **Process events** - are events defined by a process (e.g. the database system or GUI frame work). Process events reflect process states rather than instance states. Typical process events are events as an instance has been updated, created or deleted.

Instance and time events are those, which are typically defined by subject area experts in the terminology model. Process events are a more technical issue and are, usually, managed by IT experts.

Event feature relations may refer to rules as pre- and post-condition, which is one way of defining relevant state transitions [UDT]. In this case, an event happens, when both, pre- and post-condition become true.

### 3.2.3 Causality



**Causality** is a feature that describes an event (cause or reason), which activates an action (rule), i.e. a reaction describes the relationship between event and action. Causalities can be described best by subject matter experts and are an important part of the terminology model (only experts know, what will happen on the stock market, when a government changes).

### 3.3 Item

Considering object related concepts as a container containing information of any complexity (elementary value, instance, collection or classified collection), corresponding data concepts can be described on different levels:

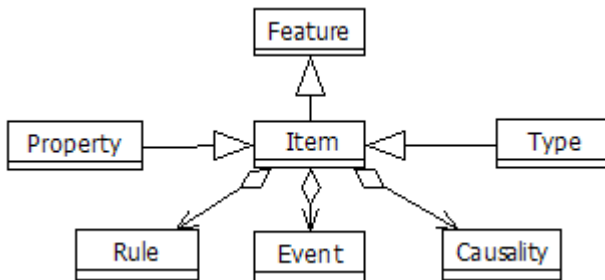
- Elementary type
- Property
- Object type
- Classification
- Aggregation types

Those categories describe **object related concepts** or schemata for data states. There is no common designation that describes the generalization of object related concepts. Here, it will be called item.

An **item** is a feature that defines common features for object related concepts as behavior and causalities. In general, each object related concept may define behavior and causalities for the object defined by the item.

An **individual item** is the reflection of an individual object property or object collection. Individual items are defined in detail in [UDT]. A **general item** (or item) describes the conceptual view to data and appears as property, elementary and object type or aggregation type.

Each item consists of a number of features, which describe the details of the item. The item defines what a concept has, how it behaves and how it reacts. The concept of an item is provided as textual definition that defines what the item is about, it's meaning.



Different categories of items have been discovered so far and are supported in TM. Items are not defined as such, but as type, property or classification.

Since objects or object views the item refers to,

are always defined in a higher context, each item has a parent item. Indeed, there are no items existing outside any context, i.e. each item (except perhaps the top-most, e.g. the universe) has a parent (item) describing the context in which the item and its features are defined.

Relational data modeling [ERM] does not consider items and defines a number of tables without defining the higher item. E.g. *invoices* could be such a table, which

does not at all mean all the invoices in the universe, but the invoices for the company, only, which deals with those invoices.

Indeed, the invoices are a feature of a company in this case. Most data models try to model "global" data sets ignoring the parent item, which is not a big problem, since the application works within the implicitly defined item, only.

### **Rules**

Rules defined for an item describe the way an object, object collection or part of an object behaves. Mainly, rules are defined in order to derive information from other properties, in order to react on different events or for defining specific behavior of related objects. Rules are often referred to by feature relations, e.g. in causalities (reactions) or constraints.

### **Constraints**

Constraints allow defining sort of validation rules for instances (object, property, collection), e.g. the *birth date* of a *person* must not be greater than the *current date* or *persons* younger than *18 years* cannot be *married* (at least in some countries).

### **Events**

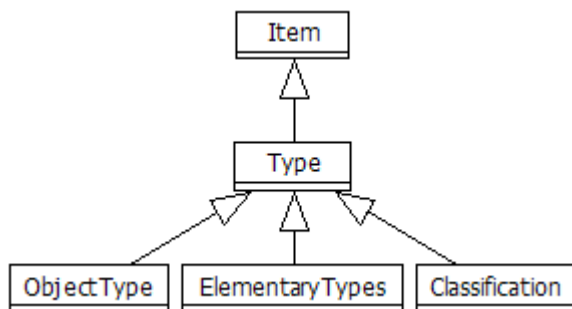
Events are often considered as process events but happen also as state transition events [UDT] caused by state transitions of one or more item instances. Events might also be defined in combination with time events.

### **Causalities**

Causalities (reactions) describe the way an item reacts on specific events. Typically, items react on process events as inserting or removing an object instance (parts, associations) or updating a property value (characteristics). When an event has been recognized, the action referred to by the causality will be called. Actions are typically defined by means of rules.

### 3.3.1 Type

A **type** is an item, which includes object types as specialization. Types are referenced by properties in order to define the way, property instances are represented. Types are also referenced (as object types) by classifications and categories.



Types do not support special features or feature relations. Those are defined for specialized types as object type and classification.

As an item, the type supports defining behavior and reactions

of the general object as rules, events and causalities. These are typical features for object types, but also elementary types and classifications may define specific behavior.

#### 3.3.1.1 Elementary types

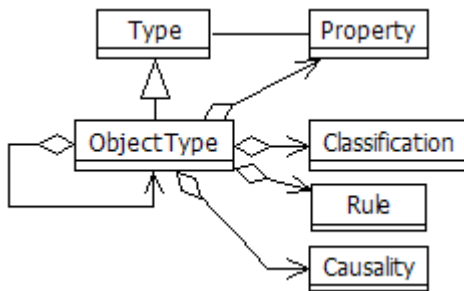
**Elementary types** are types that do not consist of properties or categories, e.g. types, which are considered as "known" by everybody or as kind of atomic. Typical elementary data types in a terminology model are text or number, which are considered as atomic from a conceptual point of view.

Elementary types are associated with certain kind of behavior, which is often not explicitly defined but expected to follow common and well-known rules defined for the elementary type (e.g. numerical operations for numbers). Similar, in some cases kind of default causalities are implicitly defined (e.g. exception as reaction on zero division).

It is a matter of view, what is considered as elementary type. In the "Terminology Model Reference" some proposed elementary types are listed.

#### 3.3.1.2 Object types

An **object type** is a type that defines the intensional aspect of a single or a set of individual objects, i.e. the properties an object type consists of and other common features of individual objects belonging to the object type. Object type definitions provide an intensional view to individual objects.



Since an object type usually refers to common features for a set of objects, object type definitions reflect a specific interest or a specific view to objects in the collection, i.e. the object type describes the properties and features of object instances as specific view to objects of the given type..

Besides properties, the object type definition may include other relevant

features as rules, subtypes, classifications etc., which are listed below.

Object types support a number of features, which provide a detailed object type definition. Most object type features are optional. The only mandatory feature required for each object type is at least one property (which might be a generalization).

Most important features are data model relevant features as

- Properties
- Keys
- Extensions

Behavioral features are described as rules, which might be referenced as constraint, condition or action.

Several features describe causalities:

- Events
- Causalities

Some models consider generalization as feature relation for object types. Here, generalization is considered as property, which is a more specific feature relation.

### Properties

Object type properties define the properties of an object type, i.e. characteristics and concept relations, which define the specific view to an object by creating an object instance as an image of reality. Properties are available for each object instance of the given object type and determine the object instance state [UDT]. Object type properties are usually defined as generalizations, characteristics, parts or associations, which provide more specific property definitions. Since properties defined in generalizations are inherited from the object type, those need not to be redefined in each specialization. Properties except generalizations might be redefined (specialized) in specialized object types.

### Fixed characteristics

**Fixed characteristics** are characteristics with an assigned value (*women* are *persons* with a fixed characteristic *sex*, which is *female* for all women). Fixed char-

acteristics overload (specialize) characteristics with the same name defined in on of the generalizations of the object type.

When a specialized object type is defined by the value of a classifying characteristic, the fixed characteristic has to be defined as property of the object type. Fixed characteristics have to refer to classifying characteristics of one of the generalizations.

### **Keys**

Keys are projections of properties (usually characteristics) from the defined object type. Keys are defined in order to identify object instances or in order to give more weight to a number of attributes.

### **Classifications**

In order to define a typed classification, i.e. a classification that may apply on objects of a given type, only, the classification may be defined as object type feature, rather than as terminology model feature, which provides more global classifications.

### **Extensions**

Extensions describe object instance collections. Actually, extensions are properties of a higher context (terminology module the object type is defined in) and should be defined as properties there.

It is a matter of viewpoint, whether extensions are considered as feature of the object type or as object type property in a higher context. On one side, it seems, that discussing the concept of an object type (e.g. invoice), extensions seem to belong to the definition of the object type. On the other hand, one could argue, that collections always require a specific context (e.g. an enterprise), for which the collection is of relevance.

For practical reasons, the terminology model also allows defining extensions as object type feature. Thus, one might define *paid* and *unpaid invoices* as feature of the object type *invoice*, as well as an object collection containing *all invoices*.

### **Local object types**

Within the context of an object type, local object types might be defined, which are known within the context of the object type, only. Features defined for the object type, only, should reference local object types. Object types referred to by other features of other object types must not be defined as local object types.

It makes a big difference, whether a property (e.g. address) or a local object type (address) has been defined. A local object type never carries data but requires at least one feature (property), referring to it.

This is an enhanced feature, which looks a bit technical. In fact, there are situations, where this becomes very useful in sense of terminology, e.g. when defining

different object types for *address* instances in *person* and *enterprise* object types. Alternatively, one could, of course, create object types *company address* and *person address* in the terminology model, but this may conflict with the way, language is used in the subject area.

### **Feature inheritance**

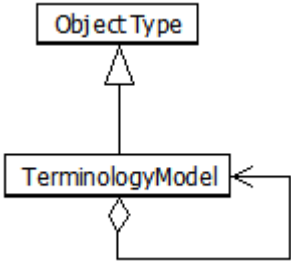
When an object type contains generalization properties, it will inherit all features from its generalizations. Features might be overloaded by redefining a feature with the same name and different meaning.

Properties, reactions, constraints and behavior might be redefined for specialized features. Redefining a feature means that the new feature replaces the feature defined with the same name for the generalization without changing the feature category.

Property specialization is a typical way in human language for expressing slight modifications for properties and behavior in specialized object types. Most technical environments do, however, not yet support property specialization.

### 3.3.1.2.1 Terminology model

A **terminology model** is an object type that describes a complex subject area, which relates to an individual or general object. When the subject area is complex, it is usually divided into sub areas, which are reflected as subordinated terminology models. In contrast to other object types, terminology models define a concept system.



A concept system as being defined in [1087] is a "set of concepts structured according to the relations among them". This corresponds very much to terminology models in TM. A concept system describes concepts used in the specific subject area. Explicitly

defining a concept system might be a good starting point for defining a terminology model, but it is too general in order to reflect important principles of expressing knowledge in human language. Thus, the concept system has to be developed later on to a terminology model.

A terminology model also defines the object type to which the knowledge area to be described, belongs. This might be either an individual object (as a specific enterprise) or a general object (enterprises as such, represented by a specific view to enterprises). Thus, the terminology model for an accounting system for an individual company will describe the company as an individual object. Providing a generic solution, which might be used by any company, the application describes the general object company from the view of accounting processes.

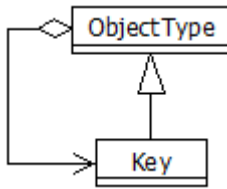
In contrast to object instances, which are defined by an object type with well-defined features, model features are more flexible and may easily change depending on specific requirements. Practically, creating a new object collection means adding a new property to the model instance, but this is not considered as significant model modification, which probably will require database reorganization.

Each terminology model should be consistent in a way that it does not refer to types, which are not defined in the terminology model itself or one of its parents.

In [ERM] the entry points are tables, which are features in the context of a project or database. The project or database does exist, but is not defined as upper context. Similar, in the object model [ODMG], where technically different contexts are provided in terms of schema, module, namespace and type, the conceptual entry points are types and extents (properties).

### 3.3.1.3 Keys

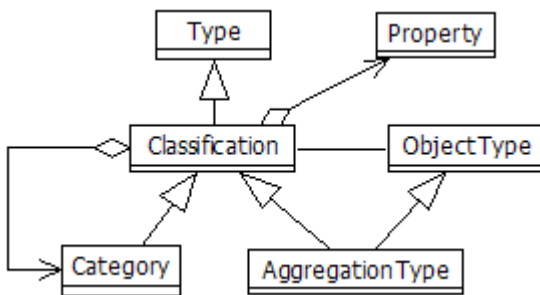
A **key** is an object type defined in the context of an object type. A key consists of a subset of properties defined for the object type owning the key. Thus, a key also defines a projection rule for object instances of the object type. A key consists of one or more key components (property relations). A **key component** is a feature relation that refers to properties defined for the object type that owns the key. A key may consist of any number of key components, but requires at least one.



At the first glance, keys seem to be an implementation issue, but this is not always case. Since keys are used for identifying object instances or ordering object instances in an object collection, keys have got a conceptual meaning, too.

### 3.3.1.4 Classifications

A **classification** is a type that describes a method, which divides a set of objects completely into distinct subsets by means of categories. Hierarchical classification might be defined by defining sub categories for each category etc. **Well-formed classifications** do have the same nesting level for each category.



When a classification refers to a type or is being defined in the context of an object type, the classification is called **typed**

**classification**. Typed classifications can apply on objects of the type associated with the classification, only.

Classifications defined as features of a type are considered automatically as typed classifications.

Classifications are referenced as type for characteristics (properties) in order to define classifying characteristics. By assigning categories to individual object, objects are associated with this category.

In order to define additional characteristics for categories of the classification, the classification may contain to a number of properties (characteristics).

In [1087], there is a limited support for categories (type of characteristic). [ODMG] supports simple classifications as enumeration. [ERM] does not support classifica-

tions, but those might be implemented as tables and table relations. [UML] supports classifications as enumerations. [UDT] considers classifications as one way for providing a P3 database schema.

### **Categories**

A classification supports a list of categories. The set of categories is supposed to divide a given object collection completely into distinct subsets.

### **Properties**

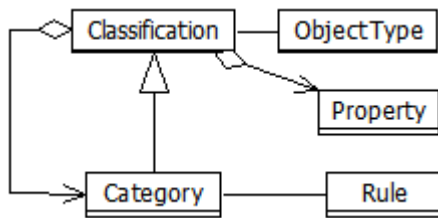
A classification may define a number of properties (characteristics) for its categories. Each property defined for the classification becomes a value in the subordinated categories.

### **Object type**

Classifications may refer to an object type, which describes the kind of objects the classification may apply on.

### 3.3.1.4.1 Category

A **category** is a concept (classification) that defines a generic subset for an object collection of a given type. The category defines the extension (borderlines) for the subset and/or a constraint.



In general, categories are classifications, i.e. a category may consist of a number of sub categories, which allow further division of object collections. When the category's classification refers to a type (**typed category**), it may define an object collection containing a set of specialized object instances.

In order to assign objects to a category, category values may be assigned to classifying characteristic (e.g. assigning the category *male* to the *person* characteristic *gender*). Alternatively, rules might be referenced that define conditions for assigning individual objects to a certain category.

Object type and properties of the category's classification are inherited from its parent classification. When the category refers to another object type then the parent classification, the category's object type must inherit from the parent's classification object type.

Properties are inherited from the upper classification and may be extended by additional properties for sub categories. Thus, each category supports all properties, which have been defined in the classification hierarchy.

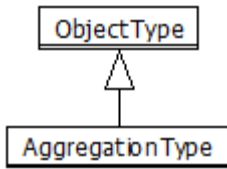
In [1087], categories are defined "type of characteristic". [ODMG] supports categories as enumerators. [ERM] does not support categories on model level. [UML] allows defining categories as enumeration literal.

#### Extension

The extension describes the object set, which is defined by the category. There are many different ways for defining the extension, but typically, it is defined by describing borderlines. E.g. persons, that earn between 2000 and 3000 EURO per month.

### 3.3.1.5 Aggregation type

An aggregation type is an object type, which inherits from a classification. In order to provide aggregated values when being applied on a collection, the aggregation type defines one or more (aggregation) properties (characteristics). Aggregation types may inherit from typed classifications, only.

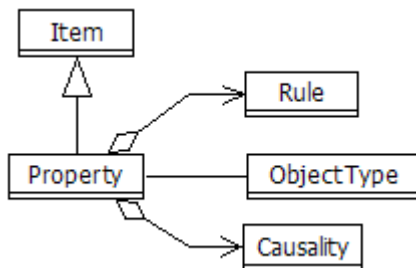


Aggregation properties usually contain aggregated data when applying on an individual object collection containing objects of the classification's object type. Aggregation properties define the aggregation rule (derivation rule) as method of the classification's object type.

Each instance of an aggregation type is identified by a category of its classification.

### 3.3.2 Property

A **property** is an item that defines the extensional aspect of a general object collection (e.g. everything that *costs money* or *persons* that have been in a *movie* at



a certain *time*, but also the *names* of a *person*). Potentially, properties define object collections even though, in many cases, those collections consist of exactly one value (e.g. person's birth date). A property is related to a value (property instance), which is part of an object instance reflecting an individual object. Properties may refer to single or complex values as well as to col-

lections of object instances. A **property instance** is a value reflecting the property for an individual object, the value of the property in a related object instance [UDT].

A property may define a collection of object instances of a given type and with a specific role (e.g. *children* of a *person*). By means of property definitions, knowledge about groups of objects can be expressed in different ways. Thus, property definitions mainly express the extensional aspect of a general object in a specific context (object type). The intensional aspect is expressed by means of an object type the property refers to, which is the object type for all object instances in the collection.

Properties can be defined as features of an object type, and thus also as features of a terminology model. Properties always are defined as features of an object type, i.e. property instances are part of an upper object instance, which represents the state of an individual object.

Considering tables in the relational model [ERM] or extents in the object model [ODMG], which are properties in the sense of the definition above, it seems that properties may exist also independent properties. However, the context for those properties often exists as individual object, but it might not be reflected as such in data modeling. In general, a property is a feature, which describes a general object collection in a higher context (object type, terminology model).

A property refers to an object type (feature relation) of the elements, which might be stored in the property instance. In order to be consistent, types referred to by the properties must be elementary (integer, text etc.) or have to be defined within the terminology model, too (object type, classification).

### **Type relation**

An important feature of the property is the type relation. Each property refers to a type, which might be an elementary type as text or number, an object type defined in the terminology model or a classification (enumerations).

### **Keys**

In order to support object rating, unique constraints or for any other conceptual reason, the property may refer to a number of keys defined for the object type referred to by the property. Typically, keys are used in connection with collection properties.

### **Derivation rule**

Properties may present the state of an object instance, but also refer to derived information. In order to define derived properties, an operation rule might be referenced, which defines, how the derived information has to be evaluated. It does not make sense storing e.g. the *age* of a *person*, but rather the *birth date*. Since *age* might be a property of conceptual interest, *age* could be defined as derived property calculated from the current date and the *birth date*.

Derived properties might be even more complex and may define derived views to objects or even create derived object instances or collections.

Practically, derivation rules might be defined directly instead of referring to a named rule. Only, when the rule becomes very complex or is rather common, an explicitly defined rule will be referenced, instead.

### **Set relations**

Typically, set relations for object type properties are described as supersets by referring to other properties. Properties referenced as superset might refer to properties defined in the higher context (parent item), to properties of the same or a related instance (local supersets).

In principle, set relations might become more complex and may describe complex set hierarchies. This is, however, not very typical for object type properties.

## **3.3.2.1 Property roles**

Within a terminology model, properties may play different roles:

- Object type properties - Properties for known for each object instance of a given object type
- Fixed characteristics - Properties, which fixed values in specialized object types
- Optional properties - Properties, which might be associated with any object type

Object type properties and fixed characteristics are defined as properties of object types. Optional properties are defined as properties of a terminology model.

### **Object type properties**

Object type properties describe a static view to an object, i.e. they describe relevant information to be provided for an object instance of the given object type (e.g. *name*, *given names*, *birth date* for object type *person*). Each object instance of a given object type provides property instances for the defined properties.

### **Fixed characteristics**

When one or more generalizations have been defined for an object type, fixed characteristics are a mean for defining specialized object types. E.g. the object type *woman* with a generalization *person* may define a fixed characteristics *sex*, which gets the value *female* for all object instances of object type *woman*. In this case, the fixed characteristics *sex* for *woman* will redefine the characteristics *sex* as being defined for object type *person*.

### **Optional properties**

Optional properties are properties defined in the context of a terminology model, which may be associated with any object type defined in the terminology model and its subsequent models. Optional properties may be added to any type of object instance, but need not. Typical optional properties are remarks or notes, which do not really belong to any specific object type.

## **3.3.2.2 Property categories**

The terminology model has chosen a subset of property categories defined in [1087] and [UML]. For matching data modeling approaches, the following categories define specialized property types.

- Generalization is a property, that defines a more general object type
- Characteristic (attribute) is a property, that describes an elementary value
- Part or partitive relation (composite aggregation) is a property that defines an owning object relation, i.e. an owner-part relation
- Association is a property that describes an object relation

The way of classifying properties differs between [ERM], [ODMG], [UML] and [1087]. The terms used for property categories are taken from [1087] (and [UML]).

### 3.3.2.2.1 Generalization

A **generalization** is a property that defines a more general object type, from which the current object type inherits all features, i.e. features of all generalizations of an object type are considered as features of the object type itself. Generalizations always contain exactly one element in the properties object collection. The value of a generalization is the value of the generalized object instance.

Thus, on the level of object instances, generalizations describe specific sort of relationship to a more general object view. Object instances of several views may inherit from the same object instance (two employments for a *person* result in two *employee* instances, which both inherit from the same *person* instance).

In fact, generalizations have an extensional aspect and may refer to supersets containing the object instances referred to as generalization instances, i.e. generalization instances are not necessarily part of the specialized instance. Since the object type defined for a generalization refers to a different object view, generalizations in general define separate object instances.

In [1087] generalizations are defined as generic object relations. [ODMG] and [UML] support generalizations but not as properties. [ERM] does not support generalizations at all.

### 3.3.2.2.2 Characteristic

A **characteristic** is a property that refers to an elementary type (number, text date, time etc.) or classification. Characteristics may refer to single or multiple elements (e.g. *given names*). Supersets for attributes might be defined as value domains, which may refer to another property or to a classification.

Characteristics can be divided into several subcategories depending on the purpose of the characteristic:

- Identifying characteristic - are used for identifying instances of an object type in an object collection
- Classifying characteristic - allow classifying an object instance according to the classification or superset the characteristic is based on
- Quantifying characteristic - provide quantities as object characteristic
- Aggregation characteristic - contains aggregated data for an aggregation type
- Descriptive characteristic - provides textual information for an individual object

The concept of characteristics restricts characteristic instances to elementary values. Data modeling also supports complex attributes [ODMG], which are conceptually considered as parts. The way of implementing complex attributes is, however, not of interest for the terminology model.

When a characteristic refers to a classification as type, the classification defines the value domain (set of permissible values) for the characteristic.

### 3.3.2.2.3 Parts

A **part** is a property that defines a partitive or owning relation to one or more object instances of a given object type. In [1087] parts are referenced as partitive concept relations. The object model [ODMG] considers part-of relations as relationships or complex attributes. [UML] supports partitive relations as composite aggregations. [ERM] defines partitive relations as table relations but not as instance relations.

Technically, it is not possible to draw a clear borderline between partitive and associative concept relations or characteristics. Conceptually, it makes a big difference, whether parts of an object instance are considered to be dependent on or owned by the referencing instance or not. It depends, however, on the specific view, whether one considers e.g. *wheels* as part of a *car* or as association between *car* and *wheels*.

Technically, part properties might be implemented as parts, complex attributes or as relationships, depending on the features supported by the target system.

#### 3.3.2.2.4 Associations

An **association** is a property that describes an object relation to other objects. In [1087] associations are defined as associative object relations. [UML] defines associations as such. The object model [ODMG] considers associations as relationships. [ERM] defines associations as table relations but not as property relations.

Similar to attributes, defining supersets for a singular or multiple associations allows defining ad hoc classifications for associations.

## 4 Terminology Model Reference

This reference manual provides object type defined for the terminology model (TM). The alphabetical object type list defines the details for all object types discussed in the previous chapters.

## 4.1 Terminology model rules

**TM** The document provides a detailed documentation of the terminology model.

The terminology model is based on definitions in „[Terminology Model II](#)“, the terminology standard ISO 1087 and common requirements on object databases (The object Data Standard: ODMG 3.0) and UML (OMG Unified Modeling Language™).

**Structure** For each terminology model or module included in the documentation a separate chapter has been provided. This chapter contains object type definitions in the first part, terminology model property definitions (extents) in the second part and classification definitions in the third part.

Object types Object types are described as shown below:

### **Name - Title**

*Conceptual definition of the object type ...*

#### **Generalizations**

**G-Name** Title  
*Definition for generalization*  
→ Object Type

...

#### **Characteristics**

**C-Name** Title  
*Detailed definition for characteristic*  
→ Type

...

#### **Parts**

**P-Name** Title  
*Detailed definition for partitive relation (part of)*  
→ Object Type

...

#### **Associations**

**A-Name** Title  
*Detailed definition for association*  
→ Object Type

...

**Module/model** For terminology modules and models properties (extents) defined on this level will be listed. When optional properties have been defined, a list of optional properties will be included.

**Name - Title**  
*Conceptual definition of the terminology model or module ...*

**Extents**

**E-Name**                      *Title*  
   *Definition for extent*  
   → *Object Type*

...

**Optional properties**

**O-Name**                      *Title*  
   *Detailed definition for optional property*  
   → *Type*

...

**Classifications**              Classifications are described as shown below:

**Name - Title**  
*Conceptual definition of classification ...*

**Categories**

**C-Name**                      *Title*  
   *Definition for category (type for typed categories, only)*  
   → *[ Object type ]*

...

**Notes**                              Notes are displayed with this yellow background. Notes should be considered as sort of comment or further explanation and are not considered as relevant part of the document.

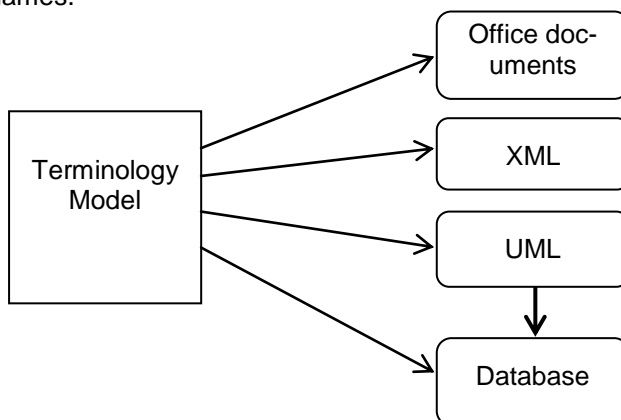
**Examples**                              Examples are presented as shown here. Examples provide more explanation for the defined concept.

## 4.2 From terminology model to database

The goal of building a terminology model are detailed definitions for a subject field, but also a base for building a data model based on the terminology model. According to our experience, subject matter experts are satisfied with terminology model definitions, since the method requires clear and detailed definitions and it reflects the role of related concepts in different scopes (class concepts).

Moreover, IT technicians benefit from terminology models, since the terminology model provides a structure that can be easily transformed into a data model or database schema.

One of the important ideas for using a terminology model as base for a data model is, that users can use designations defined in the terminology model for accessing data in the data base or in an application. Unfortunately, databases usually do not accept names used in the terminology as class or property names in the database. When the database is not able to handle conceptual names, a semantic interface has to be provided that does the mapping between technical and conceptual names.

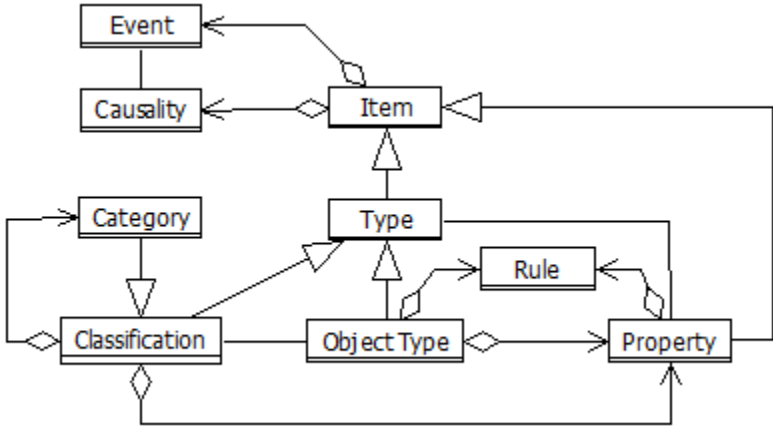


Creating an object model from the terminology model is not a big problem and requires formal changes as removing or replacing spaces in names. Thus, one may easily create an UML definition from the terminology model and use this for further processing.

Terminology-oriented systems (e.g. ODABA) allow directly generating data models from a terminology model. Moreover, one may create documents, UML presentations, application assistance (online help) and other formats from the terminology model created.

### 4.3 Terminology Model Object Types

The model definition documents details of the terminology model. The model contains conceptual relevant object types and characteristics, but no implementation specific definitions. In order to implement a terminology model, several extensions might become necessary.

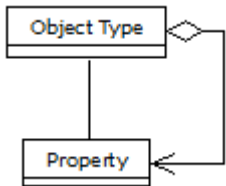


The goal of a terminology model is knowledge presentation. The terminology model covers the knowledge aspects discussed in "Terminology Model Overview" and tries to present those in a structured way. The terminology model tries to refer as much as possible to available standards, but for providing a structured knowledge presentation, it needs further extensions.

Those aspects are important for improving definitions but also for matching IT requirements.

The terminology model may be divided in different parts, which refer to different phases when developing a terminology model. Each part may be described on different levels (core level and extended level). Static terminology models describing the core level may solve about 90% of the requirements. Extended terminology models for defining behavior and causalities are more difficult, but provide a more sophisticated conceptual description of a subject area.

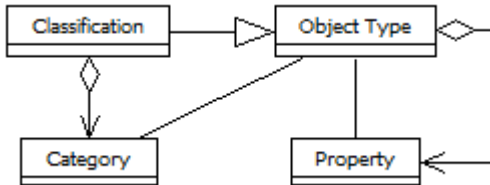
#### Static terminology model



model.

The static terminology model mainly describes object type/property/object type relations. Relevant concepts of an expert area are considered as object types, which have properties. Properties refer to types again, which have to be described by its properties etc. Such static models are typically the starting point when creating a terminology

Even though this is a very simple definition of the world, the static terminology provides a good conceptual definition of many relevant concepts on the one hand, and the base for generating database models and other technical information on the other hand.

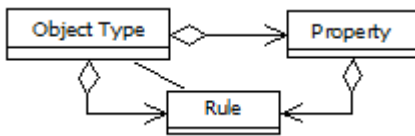


A small extension of the static terminology model includes classifications, which are often necessary for defining categories of object classes defined in the model.

The properties in the model are described according to its specific categories, since it makes a relevant difference, whether a *company* is a *person* or has *person*. In the case a *company* is a *person* the *company* inherits all the properties that are defined for the *person*.

### Behavioral terminology model

The behavioral terminology model becomes necessary, when going to develop an application. Nevertheless, the behavioral aspect is a conceptual aspect which is



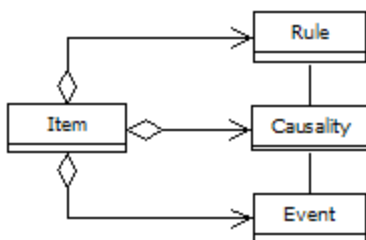
known mainly by the experts. Thus, the terminology model provides the behavioral part for describing behavior of objects of a given class. The semantics follow the common rules, i.e. more special object

types (concepts) inherit the behavior from more general object types (concepts).

Since rules or behavior may generate an output (result), rules might refer to an object type, which defines the output. Behavior or rules are usually defined for objects belonging to a specific object type. The terminology model allows describing affected objects (side effects) of a behavior, but there is no explicit support for describing cooperative behavior, i.e. behavior with more than one object as actor. Defining composite object types or parameters, which may influence the behavior, may solve this problem.

### Dynamic terminology model

The dynamic terminology model describes causalities by means of events and actions. Causalities are described as complex state transitions or time (cause or event) and a reaction (behavior or rule), which is the consequence of the event.



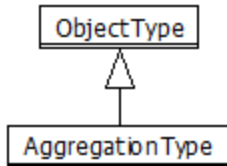
Even though, causalities are often not supported by implementation tools, they describe rather complex mechanisms for a subject area.

Special causalities are those, which are based on process events, i.e. generic events generat-

ed by the process e.g. when an object or property instance has been updated.

### 4.3.1 AggregationType - Aggregation type

An aggregation type is an object type, which inherits from a classification. In order to provide aggregated values when being applied on a collection, the aggregation type defines one or more (aggregation) properties (characteristics). Aggregation types may inherit from typed classifications, only.



Aggregation properties usually contain aggregated data when applying on an individual object collection containing objects of the classification's object type. Aggregation properties define the aggregation rule (derivation rule) as method of the classification's object type.

Each instance of an aggregation type is identified by a category of its classification.

#### Generalizations

##### ObjectType

*Object type*

The object type the aggregation inherits from allows defining any number of properties. Aggregation types have to inherit from at least one typed classification or other aggregation type.

→ ObjectType

## 4.3.2 Association - Associations

An **association** is a property that describes an object relation to other objects. In [1087] associations are defined as associative object relations. [UML] defines associations as such. The object model [ODMG] considers associations as relationships. [ERM] defines associations as table relations but not as property relations.

Similar to attributes, defining supersets for a singular or multiple associations allows defining ad hoc classifications for associations.

### Generalizations

#### Property

#### *Property*

An association is a property and inherits common features from the Property type.

→ Property

### Associations

#### inverse

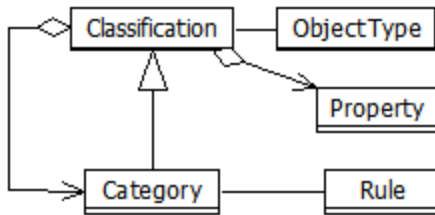
#### *Inverse association*

The inverse association is the counterpart association in the related object instance. When a car has got an *owner* (*person* association), the person might get an association *cars* for the cars the person owns. In this case, *cars* is the inverse association for *owner*.

→ Association

### 4.3.3 Category - Category

A **category** is a concept (classification) that defines a generic subset for an object collection of a given type. The category defines the extension (borderlines) for the subset and/or a constraint.



In general, categories are classifications, i.e. a category may consist of a number of sub categories, which allow further division of object collections. When the category's classification refers to a type (**typed category**), it may define an object collection containing a set of specialized object instances.

In order to assign objects to a category, category values may be assigned to classifying characteristic (e.g. assigning the category *male* to the *person* characteristic *gender*). Alternatively, rules might be referenced that define conditions for assigning individual objects to a certain category.

Object type and properties of the category's classification are inherited from its parent classification. When the category refers to another object type then the parent classification, the category's object type must inherit from the parent's classification object type.

Properties are inherited from the upper classification and may be extended by additional properties for sub categories. Thus, each category supports all properties, which have been defined in the classification hierarchy.

In [1087], categories are defined "type of characteristic". [ODMG] supports categories as enumerators. [ERM] does not support categories on model level. [UML] allows defining categories as enumeration literal.

#### Extension

The extension describes the object set, which is defined by the category. There are many different ways for defining the extension, but typically, it is defined by describing borderlines. E.g. persons, that earn between 2000 and 3000 EURO per month.

## Generalizations

### Classification

#### *Classification*

A category is a classification inheriting all features from the classification. Potentially, each category is a classification, too, which supports further subdivision for a category collection. Classification categories are, often, not explicitly named, but defined just by listing the sub categories for a category (hierarchical classification). Nevertheless, from a conceptual viewpoint categories providing sub categories are classifications. .

→ Classification

## Associations

### extension

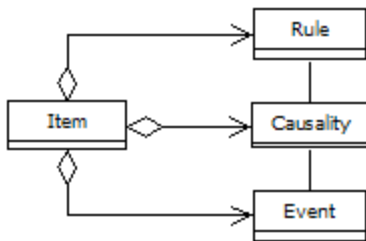
#### *Condition*

The extension describes the object set, which is defined by the category. There are many different ways for defining the extension, but typically, it is defined by describing borderlines, e.g. persons, that earn between 2000 and 3000 Euro per month.

Extensions can be defined for typed categories, only. Often, the extension is defined in terms of conditions, where the condition is a rule defined in the context of the classification's object type. In case of hierarchical classifications, the rule might also be defined in the context of the object type associated with the parent category.

→ Rule

### 4.3.4 Causality - Causality



**Causality** is a feature that describes an event (cause or reason), which activates an action (rule), i.e. a reaction describes the relationship between event and action. Causalities can be described best by subject matter experts and are an important part of the terminology model (only experts know, what will happen on the stock market, when a government changes).

#### Generalizations

<b>Feature</b>	<p><i>Feature</i></p> <p>Causality is a feature and inherits all features from the Feature type.</p> <p>→ Feature</p>
----------------	---

#### Associations

<b>action</b>	<p><i>Action</i></p> <p>The action defines the way the causality reacts on an event. Usually, the action is defined as rule in the context, which defines the causality.</p> <p>→ Rule</p>
<b>event</b>	<p><i>Event</i></p> <p>Causality refers to an event, which might be described in terms of pre- and post-conditions or as process event. Since an event is defined as set of relevant state transitions, one event is always sufficient to describe the cause or reason for a reaction.</p> <p>→ Event</p>

### 4.3.5 Characteristic - Characteristic

A **characteristic** is a property that refers to an elementary type (number, text date, time etc.) or classification. Characteristics may refer to single or multiple elements (e.g. *given names*). Supersets for attributes might be defined as value domains, which may refer to another property or to a classification.

Characteristics can be divided into several subcategories depending on the purpose of the characteristic:

- Identifying characteristic - are used for identifying instances of an object type in an object collection
- Classifying characteristic - allow classifying an object instance according to the classification or superset the characteristic is based on
- Quantifying characteristic - provide quantities as object characteristic
- Aggregation characteristic - contains aggregated data for an aggregation type
- Descriptive characteristic - provides textual information for an individual object

The concept of characteristics restricts characteristic instances to elementary values. Data modeling also supports complex attributes [ODMG], which are conceptually considered as parts. The way of implementing complex attributes is, however, not of interest for the terminology model.

When a characteristic refers to a classification as type, the classification defines the value domain (set of permissible values) for the characteristic.

#### Generalizations

##### Property

##### *Property*

A characteristic is a property and inherits common features from the Property type.

→ Property

#### Characteristics

##### initial value

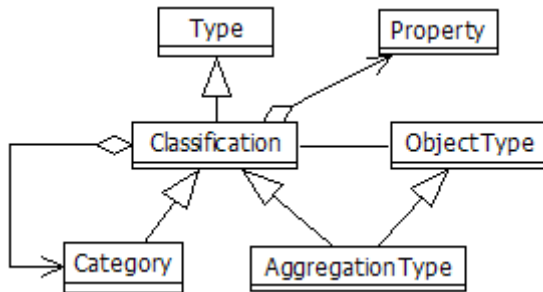
##### *Initial value*

The initial value for a characteristic is the value, which is set when creating an object instance containing the characteristic. When a superset has been defined, the value must be a valid value in the superset.

→ Text

### 4.3.6 Classification - Classifications

A **classification** is a type that describes a method, which divides a set of objects completely into distinct subsets by means of categories. Hierarchical classification



might be defined by defining sub categories for each category etc. **Well-formed classifications** do have the same nesting level for each category.

When a classification refers to a type or is being defined in the context of an object type, the classification is called **typed**

**classification**. Typed classifications can apply on objects of the type associated with the classification, only.

Classifications defined as features of a type are considered automatically as typed classifications.

Classifications are referenced as type for characteristics (properties) in order to define classifying characteristics. By assigning categories to individual object, objects are associated with this category.

In order to define additional characteristics for categories of the classification, the classification may contain to a number of properties (characteristics).

In [1087], there is a limited support for categories (type of characteristic). [ODMG] supports simple classifications as enumeration. [ERM] does not support classifications, but those might be implemented as tables and table relations. [UML] supports classifications as enumerations. [UDT] considers classifications as one way for providing a P3 database schema.

#### Categories

A classification supports a list of categories. The set of categories is supposed to divide a given object collection completely into distinct subsets.

#### Properties

A classification may define a number of properties (characteristics) for it's categories. Each property defined for the classification becomes a value in the subordinated categories.

#### Object type

Classifications may refer to an object type, which describes the kind of objects the classification may apply on.

## Generalizations

### Type

#### *Type*

A classification is a type that defines the concept of the classification and its features.

→ Type

## Parts

### characteristics

#### *Category characteristics*

A classification may define a number of additional characteristics for its categories. E.g. an education classification could add the education duration for each education category. Category characteristics must not refer to aggregated data of individual object collections.

It is an old discussion, whether categories represent data or metadata, since it appears as both. One may say, that defining categories is metadata as long as not applying to individual objects. Thus, category characteristics defined for a classification must not refer to aggregated data from individual object collections, but to data, which is independent on the set of individual objects the category applies on.

## Associations

### categories

#### *Categories*

Categories of a classification divide a set of objects (of the given object type) completely into distinct subsets.

→ Category

### object type

#### *Object type*

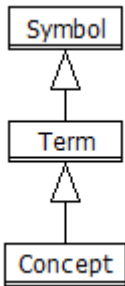
The object type describes the kind of object instances, which might be classified by the classification. The classification might be used to classify objects of the defined object type or a specialization of this type. When the classification allows classifying anything, no object type needs to be defined.

When a classification is defined as feature of an object type (local classification), it may refer to a local object types of the object type. Otherwise, the object type owning the classification is assumed to be the classification's object type.

→ ObjectType

### 4.3.7 Concept - Concepts and terms

An essential point of transferring knowledge is common understanding and referring to proper and well-defined terms.



Transferring knowledge, i.e. communication, is based on sequences of symbols. A **symbol** is any kind of signal (visual, acoustical etc.) with an agreed underlying meaning passed between sender(s) and receiver(s). A **term** (single word or sequence of words) is a symbol based on a sequence of letters and numbers. The meaning of a term is not necessarily unique but it becomes unique in a certain context. TM considers terms only as mean of expressing knowledge. Other symbols as pictures or sounds are not considered in TM.

Different terms may have the same meaning (**synonyms**). Synonyms might be defined when being used within a subject area, but also, when terminology changes. In the latter case synonyms allow mapping old terms to newer terms.

Terms are used for referring to concepts. [1087] defines a concept as "unit of knowledge created by a unique combination of characteristics". TM needs to generalize this definition slightly. Here, a **concept** is a unit of knowledge which is referred to by a term and which is defined by textual description. Thus, a concept is a term with a meaning that reflects a part of reality or an idea, but concepts do not necessarily refer to characteristics.

Simply said, any term that has got a definition or explanation is considered as concept in TM. There are no formal criteria to measure the quality of the definition or explanation. Thus, it depends on the TM developer, how good a terminology model finally is.

As mentioned in the previous chapters, concepts applying to objects may appear as individual or general concepts. TM mainly considers general concepts, although the terminology model itself may refer to an individual object (e.g. when defining the terminology model for a specific enterprise).

#### Characteristics

**definition**

*Concept definition*

A description or definition of the named concept.

➔ Text

**designation** *Concept name*  
The designation is a single word or group of words that identifies the concept within a given scope (terminology model, object type, classification).  
→ Name

**importance** *Importance of the concept*  
The importance defines the value of a concept. Typically, the terminology model distinguishes between problem relevant and technical concepts.  
→ ImportanceLevels

## Parts

**document references** *Document references*  
Document references provide links to internal or external documents containing more information about the concept.  
→ DocumentReference

**example** *Examples*  
One or more examples describing the defined concept.  
This is an example for an example.

→ Text

**provenance** *Provenance*  
Contains remarks about the ownership and history of the concept.  
→ Text

**rationales** *Rationales*  
Allows describing the reasons and the motivation for defining the concept.  
→ Text

**source** *Concept source*  
Source of the concept definition when being defined in another place or area.  
→ Text

**synonyms** *Synonyms for concept*  
List of synonyms that can be used instead of the concept name.  
→ Text

**use cases**

*Use cases for the concept*

In order to explain the concept more detailed, one or more use cases might be described.

➔ Text

### 4.3.8 DocumentReference - Document reference

Document references might be defined, when a concept is defined or explained more detailed in an external source (document or web site).

#### Characteristics

<b>AUTOIDENT</b>	<i>Internal number</i> Document references get an internal unique number for having a unique key among all document references. → Number
<b>author</b>	<i>Author or list of authors of the document</i> → Name
<b>id</b>	<i>Reference ID</i> The reference identification is a unique string, which is used in other documents to refer to this document, e.g. run99.1. Usually, document references are displayed in brackets like [run99.1]. Identifiers are limited to 10 characters. Since documents might be referenced in different places, reference ids must be unique.  It is suggested to use an acronym for the author, publishing year (and number, if necessary). → Text
<b>name</b>	<i>Semantic document name</i> This is a short name or title for the document to be displayed, when the document is referenced. The name is not language dependent. → Text
<b>path</b>	<i>Complete document path</i> Complete path to the location, where the document is stored. This can be a local path but also an URL referring to a location in the WEB. → Text
<b>publishing date</b>	<i>Publishing date</i> → Text

**publishing event**      *Publisher or conference*  
This includes the name of the publishing company or conference or event, where the document has been published.

Addisson Wessley

Object world

→ Text

**publishing place**      *Publishing place*  
This is the place (city, country or state), where the document has been published.

Berlin

Germany

→ Text

**title**      *Document title*  
The document title is usually the official title, under which the document has been published.

→ Text

**type**      *Document type*  
The document type is required, when the extension in the document name does not allow determining the document type. Usually the type is taken from the document extension. When defining a document type in this property, this is used rather than the document extension.

→ Name

## Parts

**abstract**      *Abstract*  
Short summary of the document content. This section is not multilingual and should be provided with the language for the document.

→ Text

### 4.3.9 Event - Events

An **event** describes a cause or reason, which may activate an action (rule). There is a difference between individual and general events. An **individual event** is something that had really happened, while a general event is an event that may happen. TM mainly considers general events.

Usually, general events are referenced from within causalities. The same general event might be referred to from within different causalities, in which case all related actions are called, when an individual event happens.

**General events** are features describing relevant state transitions. Depending on different categories of state transitions, three categories of events may be defined:

- **Instance events** - are defined as relevant instance state transitions that may cause a reaction (activating a rule), e.g. when a person has married (cause) this has to be registered (action).
- **Time events** - are defined as relevant time state transitions (e.g. a person's birthday) that require appropriate actions (as sending a mail). This does not differ in principle from object state transitions, but time plays a special role and thus, it becomes useful to consider time-dependent reactions separately.
- **Process events** - are events defined by a process (e.g. the database system or GUI frame work). Process events reflect process states rather than instance states. Typical process events are events as an instance has been updated, created or deleted.

Instance and time events are those, which are typically defined by subject area experts in the terminology model. Process events are a more technical issue and are, usually, managed by IT experts.

Event feature relations may refer to rules as pre- and post-condition, which is one way of defining relevant state transitions [UDT]. In this case, an event happens, when both, pre- and post-condition become true.

#### Generalizations

**Feature**

*Feature*

An event is a feature and inherits all features provided by the Feature type.

➔ Feature

## Characteristics

**type** *Event type*  
The event type describes the type of the selected event.  
→ EventType

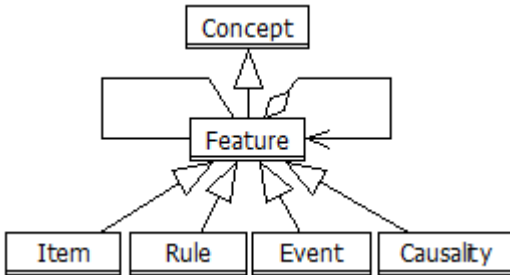
## Associations

**post condition** *Post-condition*  
State and time events require a post-condition, which refers to a constraint expressed by a rule defined in the context owning the event.  
→ Rule

**pre condition** *Pre-condition*  
State and time events require a pre-condition, which refers to a constraint expressed by a rule defined in the context owning the event.  
→ Rule

### 4.3.10 Feature - Feature categories

Many concepts refer to subordinated concepts (e.g. via concept relations). Subordinated concepts that belong to exactly one concept (parent or owner) are called **feature**. Features may appear as directly subordinated concepts (e.g. characteristics) or as concept relation referring to other concepts.



When a feature refers to another concept, the feature still belongs to its owner, but not the referenced concept. E.g. a constraint hat refers to a rule belongs to its owning concept, but not the rule referenced. Features, that define concept relations, are also called feature relations. A **feature relation** is a concept

relation that refers to a feature defined as feature of another concept. Usually, feature relations do not contain subordinated features, but the relation to exactly one other feature. There are, however, also features, that define a feature relation by referring to the feature of another concept and providing own features. Thus, TM does not strictly distinguish between feature and feature relation, but refers to feature and feature relation in order to refer to the owner or relation aspect of a feature.

The feature relation is a concept describing the use of a defined feature within another feature, while subordinated features are part of its parent feature. E.g. the validation rule (constraint) for the *salary* property of an *employee* (feature relation) refers to a rule (feature) defined for object type *employee* (checks, whether an *employee* does not earn more than his boss). The rule is defined as feature of the object type *employee* but it is referenced from the property *salary* as constraint (feature relation). Thus, the feature relation defines the role the referenced feature plays in a specific environment.

Defining the details of a concept by means of features allows expressing detailed knowledge not only about a subject area, but also about object types, properties and object collections.

Features may get subordinated features, again. Thus, features may form a hierarchy. The meaning of the term referring to a feature is uniquely defined within the owning feature, only.

Since features define the meaning of a term in the context of another feature, the same term might be assigned to features owned by another feature (e.g. proper-

ties assigned to different object types may use the same name with different meaning, as *size* property for *person* and *company*). Thus, the meaning of a feature, i.e. its concept definition, may differ depending on the owning feature.

In a terminology model, each concept is defined at least in the context of the individual or general object, which provides the frame for the terminology model. Hence, instead of defining concepts, TM defines features and feature hierarchies.

In order to simplify the terminology model, features levels are assigned to features. Level 1 features allow defining a simple terminology model and include object types, classifications and properties. Level 2 features describe an enhanced terminology model that also includes rules, causalities and keys.

In [1087], features are not defined as such. There, object relations and characteristics are defined as details of a concept, but those are not considered as subordinated concepts. In [ODMG], properties and behavior (what objects of a class are able to do) are considered as features for object types, as well as extents defining object collections or keys. In addition, TM supports a number of feature categories, which play an important role when expressing knowledge.

- Rule
- Event
- Causality
- Item

Items summarize a number of more specific features. Rule, event and causality are features defined for items as well as constraints.

Exceptions are defined as features in [ODMG], too, in order to describe, how objects of a given type behave in unexpected situations. Since exceptions seem to be rather an implementation issue than a conceptual one, exceptions are not considered as feature categories in TM.

## Generalizations

### Concept

*Concept*

A feature is a concept, which provides a conceptual definition for the feature.

➔ Concept

## Characteristics

### type

*Feature type*

Features are never defined as such, but appear always as specialized features. The feature type refers to the type of the specialized feature.

➔ FeatureTypes

### 4.3.11 FixedCharacteristic - Fixed characteristic

A **fixed characteristic** is a characteristic, which has an assigned value that cannot be changed. A typical way of defining subsets (properties) or subtypes (types) is defining one or more fixed characteristics. In order to define a subset, any characteristic might be defined as fixed characteristic assigning a value to it. Thus, e.g. for the *persons* property a subset *women* could be defined by defining the characteristics *gender* as fixed with the value *female*.

When inheriting from one or more generalizations, the specialized type could also be defined by means of fixed characteristics.

#### Generalizations

<b>Characteristic</b>	<i>Characteristic</i> A fixed characteristics is a characteristic with an assigned value. → Characteristic
-----------------------	--

#### Characteristics

<b>value</b>	<i>Value for the characteristic</i> The fixed value defines the value set for the characteristic in order to define a specialized object type or subset by means of a category. → Text
--------------	--

### 4.3.12 Generalization - Generalization

A **generalization** is a property that defines a more general object type, from which the current object type inherits all features, i.e. features of all generalizations of an object type are considered as features of the object type itself. Generalizations always contain exactly one element in the properties object collection. The value of a generalization is the value of the generalized object instance.

Thus, on the level of object instances, generalizations describe specific sort of relationship to a more general object view. Object instances of several views may inherit from the same object instance (two employments for a *person* result in two *employee* instances, which both inherit from the same *person* instance).

In fact, generalizations have an extensional aspect and may refer to supersets containing the object instances referred to as generalization instances, i.e. generalization instances are not necessarily part of the specialized instance. Since the object type defined for a generalization refers to a different object view, generalizations in general define separate object instances.

In [1087] generalizations are defined as generic object relations. [ODMG] and [UML] support generalizations but not as properties. [ERM] does not support generalizations at all.

#### Generalizations

##### Association

##### *Association*

A generalization is an association that provides a more general view to a certain object instance. Thus, a generalization is not only a property, but an association and inherits important features from the association.

→ Association

### 4.3.13 Item - Item

Considering object related concepts as a container containing information of any complexity (elementary value, instance, collection or classified collection), corresponding data concepts can be described on different levels:

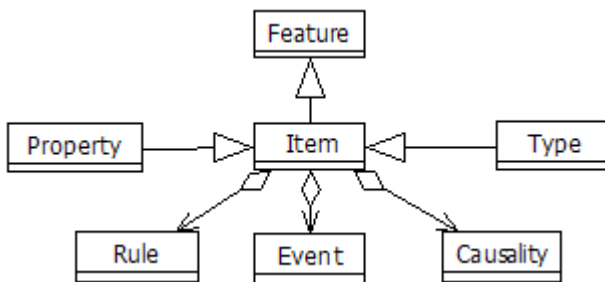
- Elementary type
- Property
- Object type
- Classification
- Aggregation types

Those categories describe **object related concepts** or schemata for data states. There is no common designation that describes the generalization of object related concepts. Here, it will be called item.

An **item** is a feature that defines common features for object related concepts as behavior and causalities. In general, each object related concept may define behavior and causalities for the object defined by the item.

An **individual item** is the reflection of an individual object property or object collection. Individual items are defined in detail in [UDT]. A **general item** (or item) describes the conceptual view to data and appears as property, elementary and object type or aggregation type.

Each item consists of a number of features, which describe the details of the item. The item defines what a concept has, how it behaves and how it reacts. The concept of an item is provided as textual definition that defines what the item is about, it's meaning.



Different categories of items have been discovered so far and are supported in TM. Items are not defined as such, but as type, property or classification.

Since objects or object views the item refers to, are always defined in a higher context, each item has a parent item. Indeed, there are no items existing outside any context, i.e. each item (except perhaps the top-most, e.g. the universe) has a parent (item) describing the context in which the item and its features are defined.

Relational data modeling [ERM] does not consider items and defines a number of tables without defining the higher item. E.g. *invoices* could be such a table, which does not at all mean all the invoices in the universe, but the invoices for the company, only, which deals with those invoices.

Indeed, the invoices are a feature of a company in this case. Most data models try to model "global" data sets ignoring the parent item, which is not a big problem, since the application works within the implicitly defined item, only.

### **Rules**

Rules defined for an item describe the way an object, object collection or part of an object behaves. Mainly, rules are defined in order to derive information from other properties, in order to react on different events or for defining specific behavior of related objects. Rules are often referred to by feature relations, e.g. in causalities (reactions) or constraints.

### **Constraints**

Constraints allow defining sort of validation rules for instances (object, property, collection), e.g. the *birth date* of a *person* must not be greater than the *current date* or *persons* younger than *18 years* cannot be *married* (at least in some countries).

### **Events**

Events are often considered as process events but happen also as state transition events [UDT] caused by state transitions of one or more item instances. Events might also be defined in combination with time events.

### **Causalities**

Causalities (reactions) describe the way an item reacts on specific events. Typically, items react on process events as inserting or removing an object instance (parts, associations) or updating a property value (characteristics). When an event has been recognized, the action referred to by the causality will be called. Actions are typically defined by means of rules.

## **Generalizations**

### **Feature**

#### *Feature*

An item is a feature that inherits common features defined for the **Feature** object type.

→ Feature

## Parts

### causalities

#### *Causalities*

Causalities describe the reaction on specific events as causality, i.e. the action executed as consequence of an event in the given context. Rules and events the causality is referring to have to be defined in the same context that defines the causality, or in one of its generalizations.

➔ Causality

### events

#### *Events*

Events refer to process events but also to instance events caused by state transitions of an item. Events might also be defined in combination with time events.

Instance events are either property or object events, depending on whether the item defines a property or an object type.

➔ Event

### rules

#### *Rules*

Rules provide definition of behavior for the item defining the rule. Any number of rules can be defined for an item in order to describe constraints, state transitions or operations.

➔ Rule

## Associations

### constraints

#### *Constraints*

Constraints allow defining sort of validation rules for items (object type, property, classification), e.g. the *birth date* of a *person* must not be greater than the *current date* or *persons* younger than *18 years* cannot be *married*.

Constraints usually define validation rules for instances of a given item and may refer to rules defined for this item. Usually, constraints refer to rules defined as item features or as features of one of its generalizations.

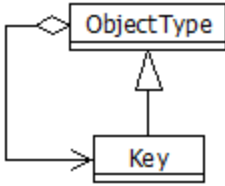
Object constraints allow defining the conditions (rules) for defining a subtype from its generalizations. This is, however, rather a rare case. E.g. *adult* inheriting from *person* (generalization) might be defined as *persons* with an *age* greater or equal than *18 years*.

In principle, it is also possible to define types based constraints. This is defined, however, rather as set relation than as a new object type. Since there is no clear borderline between type and set relation, each defined object instance collection could also be defined as new type.

Practically, constraints might be defined directly instead of referring to a named rule. Only, when the rule becomes very complex or is rather common, an explicitly defined rule might be referenced, instead.

➔ Rule

### 4.3.14 Key - Keys



A **key** is an object type defined in the context of an object type. A key consists of a subset of properties defined for the object type owning the key. Thus, a key also defines a projection rule for object instances of the object type. A key consists of one or more key components (property relations). A **key component** is a feature relation that refers to properties defined for the object type that owns the key. A key may consist of any number of key components, but requires at least one.

At the first glance, keys seem to be an implementation issue, but this is not always case. Since keys are used for identifying object instances or ordering object instances in an object collection, keys have got a conceptual meaning, too.

#### Generalizations

##### ObjectType

*Object type*

A key is an object type, which only inherits characteristics from ObjectType. All other features are disabled for keys.

➔ ObjectType

#### Associations

##### components

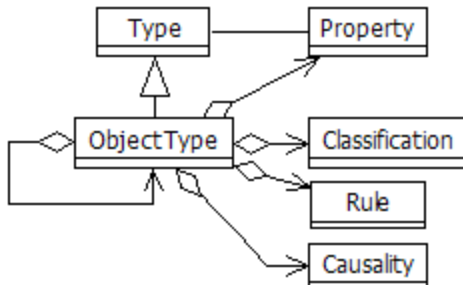
*Key components*

Key properties or components are a feature relations, which define the components a key consists of. Usually, key components are characteristics, but parts might be defined as key components as well. Associations or generalizations cannot be defined as key components.

➔ Property

### 4.3.15 ObjectType - Object types

An **object type** is a type that defines the intensional aspect of a single or a set of individual objects, i.e. the properties an object type consists of and other common features of individual objects belonging to the object type. Object type definitions provide an intensional view to individual objects.



Since an object type usually refers to common features for a set of objects, object type definitions reflect a specific interest or a specific view to objects in the collection, i.e. the object type describes the properties and features of object instances as specific view to objects of the given type.

Besides properties, the object type definition may include other relevant features as rules, subtypes, classifications etc., which are listed below.

Object types support a number of features, which provide a detailed object type definition. Most object type features are optional. The only mandatory feature required for each object type is at least one property (which might be a generalization).

Most important features are data model relevant features as

- Properties
- Keys
- Extensions

Behavioral features are described as rules, which might be referenced as constraint, condition or action.

Several features describe causalities:

- Events
- Causalities

Some models consider generalization as feature relation for object types. Here, generalization is considered as property, which is a more specific feature relation.

#### Properties

Object type properties define the properties of an object type, i.e. characteristics and concept relations, which define the specific view to an object by creating an object instance as an image of reality. Properties are available for each object instance of the given object type and determine the object instance state [UDT]. Object type properties are usually

defined as generalizations, characteristics, parts or associations, which provide more specific property definitions. Since properties defined in generalizations are inherited from the object type, those need not to be redefined in each specialization. Properties except generalizations might be redefined (specialized) in specialized object types.

### **Fixed characteristics**

**Fixed characteristics** are characteristics with an assigned value (*women are persons* with a fixed characteristic *sex*, which is *female* for all women). Fixed characteristics overload (specialize) characteristics with the same name defined in one of the generalizations of the object type.

When a specialized object type is defined by the value of a classifying characteristic, the fixed characteristic has to be defined as property of the object type. Fixed characteristics have to refer to classifying characteristics of one of the generalizations.

### **Keys**

Keys are projections of properties (usually characteristics) from the defined object type. Keys are defined in order to identify object instances or in order to give more weight to a number of attributes.

### **Classifications**

In order to define a typed classification, i.e. a classification that may apply on objects of a given type, only, the classification may be defined as object type feature, rather than as terminology model feature, which provides more global classifications.

### **Extensions**

Extensions describe object instance collections. Actually, extensions are properties of a higher context (terminology module the object type is defined in) and should be defined as properties there.

It is a matter of viewpoint, whether extensions are considered as feature of the object type or as object type property in a higher context. On one side, it seems, that discussing the concept of an object type (e.g. *invoice*), extensions seem to belong to the definition of the object type. On the other hand, one could argue, that collections always require a specific context (e.g. an *enterprise*), for which the collection is of relevance.

For practical reasons, the terminology model also allows defining extensions as object type feature. Thus, one might define *paid* and *unpaid invoices* as feature of the object type *invoice*, as well as an object collection containing *all invoices*.

### **Local object types**

Within the context of an object type, local object types might be defined, which are known within the context of the object type, only. Features defined for the object

type, only, should reference local object types. Object types referred to by other features of other object types must not be defined as local object types.

It makes a big difference, whether a property (e.g. address) or a local object type (address) has been defined. A local object type never carries data but requires at least one feature (property), referring to it.

This is an enhanced feature, which looks a bit technical. In fact, there are situations, where this becomes very useful in sense of terminology, e.g. when defining different object types for *address* instances in *person* and *enterprise* object types. Alternatively, one could, of course, create object types *company address* and *person address* in the terminology model, but this may conflict with the way, language is used in the subject area.

### Feature inheritance

When an object type contains generalization properties, it will inherit all features from its generalizations. Features might be overloaded by redefining a feature with the same name and different meaning.

Properties, reactions, constraints and behavior might be redefined for specialized features. Redefining a feature means that the new feature replaces the feature defined with the same name for the generalization without changing the feature category.

Property specialization is a typical way in human language for expressing slight modifications for properties and behavior in specialized object types. Most technical environments do, however, not yet support property specialization.

## Generalizations

### Type

#### *Type*

An object type is a type, which supports additional features and feature relations.

→ Type

## Parts

### associations

#### *Associations*

Associations are instance properties, which define associative object relations to other object instances.

→ Association

### characteristics

#### *Characteristics*

Characteristics are properties defined in the context of an object type, which mainly define the object state of an object instance.

→ Attribute

**classifications**

*Classifications referenced by the object type*

Classifications may be defined in the context of an object type (or terminology module or model), which is considered as classification owner. The meaning of classifications may differ for contexts of different types.

Classification that may apply to any type of objects should be defined in the context of the terminology model. Classifications applying to object of a specific type, only, should be defined in the context of the corresponding object type.

➔ Classification

**extensions**

*Object extensions*

Object extensions are (global) features of the terminology model or module. Thus, object extensions are not part of an object instance and context property values do not depend on specific object instance, i.e. object extensions do not influence the object state.

Theoretically, object extensions could be defined as properties in the corresponding parent item. Just because practical views tend to refer to object extensions, those might be defined as feature of the object type. In general, it is a better style, defining terminology model or module properties not as object type extensions, but as properties of the terminology model or module.

➔ Property

**fixed characteristics**

*Fixed characteristics*

When a specialized object type is defined by the value of a classifying characteristic, the fixed characteristics has to be defined as feature of the object type. Fixed characteristics have to refer to classifying characteristics of one of the generalizations.

➔ FixedCharacteristic

**generalizations**

*Generalizations*

Generalizations are instance properties, which refer to a more general view to an object.

➔ Generalization

<b>keys</b>	<p><i>Keys</i></p> <p>Any number of keys might be defined in the context of an object type. Within an object type, keys define a projection by means of instance properties defined for the type or one of its generalizations.</p> <p>➔ Key</p>
<b>parts</b>	<p><i>Parts</i></p> <p>Parts are properties describing objects tightly connected (owned) by the object instance. Nevertheless, parts refer to a number of object instances, which provide a specific view to the object referenced.</p> <p>➔ Part</p>
<b>types</b>	<p><i>Types defined in the context of the object type</i></p> <p>Within the context of an object type any number of (local) object types might be defined. Type definitions are provided, typically for terminology models and modules in order to define relevant views to objects for different subject areas.</p> <p>➔ ObjectType</p>

## Associations

<b>derivations</b>	<p><i>Type derivation rules</i></p> <p>In order to evaluate derived properties, derivation rules of any type might be provided. In contrast to property evaluation rules, type evaluation rules allow evaluating several derived properties of an object instance as well as deriving context properties.</p> <p>➔ ObjectType</p>
--------------------	---

### 4.3.16 Part - Parts

A **part** is a property that defines a partitive or owning relation to one or more object instances of a given object type. In [1087] parts are referenced as partitive concept relations. The object model [ODMG] considers part-of relations as relationships or complex attributes. [UML] supports partitive relations as composite aggregations. [ERM] defines partitive relations as table relations but not as instance relations.

Technically, it is not possible to draw a clear borderline between partitive and associative concept relations or characteristics. Conceptually, it makes a big difference, whether parts of an object instance are considered to be dependent on or owned by the referencing instance or not. It depends, however, on the specific view, whether one considers e.g. *wheels* as part of a *car* or as association between *car* and *wheels*.

Technically, part properties might be implemented as parts, complex attributes or as relationships, depending on the features supported by the target system.

#### Generalizations

##### Property

##### *Property*

A part of an object is a property, which refers to a collection of one or more object instances owned by the object. Parts inherit all features from the Property type.

➔ Property

#### Associations

##### keys

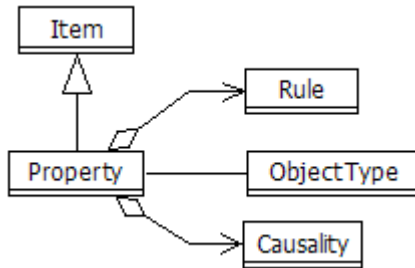
##### *Keys used by the property*

In order to identify object instances or order instances within a collection, keys defined in the object type referenced by the property might be referenced.

➔ Key

### 4.3.17 Property - Property

A **property** is an item that defines the extensional aspect of a general object collection (e.g. everything that *costs money* or *persons* that have been in a *movie* at



at a certain *time*, but also the *names* of a *person*). Potentially, properties define object collections even though, in many cases, those collections consist of exactly one value (e.g. person's birth date). A property is related to a value (property instance), which is part of an object instance reflecting an individual object. Properties may refer to single or complex values as well as to col-

lections of object instances. A **property instance** is a value reflecting the property for an individual object, the value of the property in a related object instance [UDT].

A property may define a collection of object instances of a given type and with a specific role (e.g. *children* of a *person*). By means of property definitions, knowledge about groups of objects can be expressed in different ways. Thus, property definitions mainly express the extensional aspect of a general object in a specific context (object type). The intensional aspect is expressed by means of an object type the property refers to, which is the object type for all object instances in the collection.

Properties can be defined as features of an object type, and thus also as features of a terminology model. Properties always are defined as features of an object type, i.e. property instances are part of an upper object instance, which represents the state of an individual object.

Considering tables in the relational model [ERM] or extents in the object model [ODMG], which are properties in the sense of the definition above, it seems that properties may exist also independent properties. However, the context for those properties often exists as individual object, but it might not be reflected as such in data modeling. In general, a property is a feature, which describes a general object collection in a higher context (object type, terminology model).

A property refers to an object type (feature relation) of the elements, which might be stored in the property instance. In order to be consistent, types referred to by the properties must be elementary (integer, text etc.) or have to be defined within the terminology model, too (object type, classification).

## Type relation

An important feature of the property is the type relation. Each property refers to a type, which might be an elementary type as text or number, an object type defined in the terminology model or a classification (enumerations).

## Keys

In order to support object rating, unique constraints or for any other conceptual reason, the property may refer to a number of keys defined for the object type referred to by the property. Typically, keys are used in connection with collection properties.

## Derivation rule

Properties may present the state of an object instance, but also refer to derived information. In order to define derived properties, an operation rule might be referenced, which defines, how the derived information has to be evaluated. It does not make sense storing e.g. the *age* of a *person*, but rather the *birth date*. Since *age* might be a property of conceptual interest, *age* could be defined as derived property calculated from the current date and the *birth date*.

Derived properties might be even more complex and may define derived views to objects or even create derived object instances or collections.

Practically, derivation rules might be defined directly instead of referring to a named rule. Only, when the rule becomes very complex or is rather common, an explicitly defined rule will be referenced, instead.

## Set relations

Typically, set relations for object type properties are described as supersets by referring to other properties. Properties referenced as superset might refer to properties defined in the higher context (parent item), to properties of the same or a related instance (local supersets).

In principle, set relations might become more complex and may describe complex set hierarchies. This is, however, not very typical for object type properties.

## Generalizations

### Item

### *Item*

A property defines an item and inherits several common features from **Item**.

➔ Item

## Characteristics

**derived** *Derived property*  
Derived properties are properties, which are derived from other values of an object instance. Derived values need to refer to a derivation rule defined for the property (derivations).  
→ LOGICAL

## Associations

**derivations** *Property derivation rules*  
In order to evaluate derived properties, a derivation rule might be provided. In contrast to type evaluation rules, property evaluation rules allow evaluating the value for the property, only.  
→ Rule

**fixed attribute** *Fixed characteristic*  
When a property defines a subset by category, a fixed characteristic is required, which is defined as characteristic of the object type referenced by the property. The fixed attribute has to refer to a classifying characteristics defined in the referenced object type.  
→ FixedCharacteristic

**instance type** *Instance type*  
Dependent on the complexity of the property, it may refer to an elementary type, an object type or a classification, which describe the structure of object instances referred to by the property.  
When the property defines a collection, the type defines the object type for instances in the collection. This need not to be exactly the type of object instances assigned to the property but a generalization of all object instances, which are element of the property.  
When any kind of object instance might be collected in the property collection, the most generic object type as *anything* or similar might be referred to as type..  
→ ObjectType

**keys**

*Keys*

When a property is a collection property, the collection might refer to one or more key as mean for accessing individual object instances in the collection, for providing rating sequences or in order to express unique constraints.

➔ Key

**subsets**

*Subsets*

Within a set hierarchy, any number of subsets might be defined for a property. Subsets are properties defined in a higher context.

➔ Property

**supersets**

*Supersets*

When being defined within a set hierarchy, the property may refer to one or more supersets. Object type properties may refer to only one superset. Supersets are properties defined in a parent item.

Supersets may also be defined for characteristics, in which case the superset defines the set of permissible values (value domain). As value domain a classification might be referenced as well as a collection property.

➔ Property

### 4.3.18 Rule - Rules

Object types and properties support a static definition of objects and object classes. An advanced knowledge approach, however, is to describe, how objects behave. Typically, the behavior of objects is described as common behavior of objects of a given object type ("*birds* are able to *fly*" or "*things* are able to *fall*").

One typical approach is describing behavior as rules. A **rule** is a feature that describes how object or property instances change from one state to the other or how objects interact with other objects. Rules are referred to for different purposes (rule categories):

- **Constraints** are used as validation rules for objects and properties
- State **transitions** describe the way how objects change or interact
- **Operations** provide derived information

Defining rules is an advanced approach and not typically used when starting defining concepts. Later on, it becomes, however, important, because building applications is impossible without knowing the rules, according to which objects in the application behave and interact with each other.

Typically, rules apply on single values, object instances or object collections. In order to control rules, any number of parameters (properties) might be passed to a rule. Depending on the rule category, it may change the state of the instance it applies on or return a result property. Parameters and result are features of the rule.

There is no clear distinction between rule and characteristic or property, since characteristics (like *age*) might be also expressed as operation rule, which returns the *age*. Again, it depends on the experts view, whether age is considered more as characteristic of person or as rule evaluating the age or as both.

#### Generalizations

<b>Feature</b>	<i>Feature</i> A rule is a feature and inherits all features from the Feature type. → Feature
----------------	---

#### Characteristics

<b>type</b>	<i>Rule type</i> A rule type should be defined in order to distinguish between different purposes of rules. → RuleTypes
-------------	---

## Parts

### parameters

#### *Parameter properties*

Moreover, rules may refer to a number of properties passed as parameters in order to control the behavior of the rule.

→ Property

### result

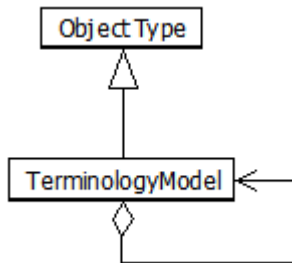
#### *Result property*

Rules may refer to a property in order to define the result returned from the rule.

→ Property

### 4.3.19 TerminologyModel - Terminology model

A **terminology model** is an object type that describes a complex subject area, which relates to an individual or general object. When the subject area is complex, it is usually divided into sub areas, which are reflected as subordinated terminology models. In contrast to other object types, terminology models define a concept system.



A concept system as being defined in [1087] is a "set of concepts structured according to the relations among them". This corresponds very much to terminology models in TM. A concept system describes concepts used in the specific subject area. Explicitly

defining a concept system might be a good starting point for defining a terminology model, but it is too general in order to reflect important principles of expressing knowledge in human language. Thus, the concept system has to be developed later on to a terminology model.

A terminology model also defines the object type to which the knowledge area to be described, belongs. This might be either an individual object (as a specific enterprise) or a general object (enterprises as such, represented by a specific view to enterprises). Thus, the terminology model for an accounting system for an individual company will describe the company as an individual object. Providing a generic solution, which might be used by any company, the application describes the general object company from the view of accounting processes.

In contrast to object instances, which are defined by an object type with well-defined features, model features are more flexible and may easily change depending on specific requirements. Practically, creating a new object collection means adding a new property to the model instance, but this is not considered as significant model modification, which probably will require database reorganization.

Each terminology model should be consistent in a way that it does not refer to types, which are not defined in the terminology model itself or one of its parents.

In [ERM] the entry points are tables, which are features in the context of a project or database. The project or database does exist, but is not defined as upper context. Similar, in the object model [ODMG], where technically different contexts are provided in terms of schema, module, namespace and type, the conceptual entry points are types and extents (properties).

## Generalizations

**TerminologyModule** *Terminology module*  
A terminology model is a terminology module and inherits the features from the terminology module.  
→ TerminologyModule

## Associations

**generalization** *Model generalization*  
When the model inherits from a (generic) model, the generalization refers to the more general terminology model. In case of model inheritance, the specialized model inherits all features from its generalization.  
Theoretically, a model might have more than one generalization, but practical, there is none or one generalization defined for a model.  
The property loads the generalization property defined in the ObjectType the terminology model is based on.  
→ TerminologyModel

**optional properties** *Optional properties*  
Optional properties in a terminology model are features, which define a number of common properties, which might be associated with any object instance.  
→ Property

**parent model** *Parent model*  
When the terminology model has been defined as sub-model, the parent model refers to the terminology model the current model is a sub-model for.  
→ TerminologyModel

**specializations** *Model specializations*  
In case of model inheritance, the generic model may refer to a list of specialized models.  
→ TerminologyModel

**sub models** *Sub models*  
In order to divide complex areas into subareas, several sub-models might be defined for a terminology model, where each sub model defines a conceptual separated subarea.  
→ TerminologyModel

### 4.3.20 TerminologyModule - Terminology Module

A **terminology module** allows collecting a set of concepts (object types and classifications) which provide a view to a certain aspect of a knowledge area - a concept field [1087]. Practically, modules are used for grouping concepts by specific categories resulting from the structure of the knowledge area.

Hence, the concept of terminological module is rather a practical issue in order to improve the use of a terminology model. From a modeling point of view, it is not required by TM. The distinction between model and module is rather based on practical experiences than on theoretical considerations. Thus, it depends on the view to the problem, whether a subordinated concept system is defined as module or model.

#### Generalizations

##### ObjectType

*Object type*

A terminology module is an object type that defines the specific view to the individual or abstract object the terminology module refers to.

→ ObjectType

#### Associations

##### causalities

*Complex causalities*

Most causalities are defined in the context of a type or property. When an event causing a reaction becomes, however, more complex, e.g. because several object instances (of different object types) are required in order to define the event, the causality cannot be described in the context of a simple object type or property.

The property loads the causality property defined in the ObjectType the terminology module is based on.

→ Causality

## **classifications**

### *Global classifications*

Usually, a classification applies to objects (instances) of a given type. Nevertheless, classifications sometimes seem to be global in the sense that they may apply to any type of object. This usually results from the fact that the object type hierarchy is usually not defined completely (e.g. a terminology model may define persons and cars without considering that both are things).

The terminology module allows defining classifications, which are considered as global classifications in the given context, which means that they may apply to any group of objects in the terminology module.

The property loads the classification property defined in the ObjectType the terminology module is based on.

→ Classification

## **collections**

### *Module collections*

Collections defined for a module represent entry points for the module. Each piece of information can be addressed starting with one of the collections defined for the terminology module. Thus, collections represent the object sets, which are considered as global in the context of a terminology model or module.

Module collections do not necessarily belong to the context instance described by the terminology model, but may refer any object collection which is of interest within the given context.

→ Property

## **parent module**

### *Parent module*

When the terminology module is a sub-module, it refers to exactly one parent module, for which it is defined as sub-module.

→ TerminologyModule

## **processes**

### *Processes*

Processes define relevant actions to be performed for the module. A process is a rule, but in contrast to other rules, a process is usually initiated explicitly.

→ Rule

## **sub modules**

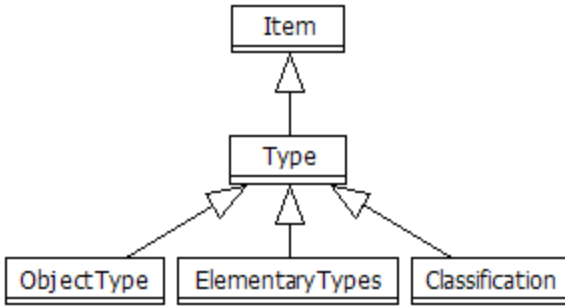
### *Sub modules*

In order to group tasks, which do not depend on the organization structure of the model instance, a terminology model or module might be divided into a number of sub-modules.

➔ TerminologyModule

### 4.3.21 Type - Type

A **type** is an item, which includes object types as specialization. Types are referenced by properties in order to define the way, property instances are represented. Types are also referenced (as object types) by classifications and categories.



Types do not support special features or feature relations. Those are defined for specialized types as object type and classification.

As an item, the type supports defining behavior and reactions of the general object as rules, events and causalities. These are typical features for object types, but also elementary types and classifications may define specific behavior.

### Generalizations

**Item**

*Item*

A type is an item and inherits common features as behavior and causalities from **Item**.

➔ Item

## 4.4 Classifications

TM provides several classifications for different purposes. Mainly classifications are provided in order to define type hierarchies (categories) for TM object types (see `FeatureTypes`). Other classifications contain proposed values for object type categories (`ElementaryTypes`).

## 4.4.1 ElementaryTypes - Elementary types

Elementary types are types as text or number, which are considered as atomic from a conceptual point of view. It is a matter of taste, what is considered as elementary type. The elementary types listed below should be considered rather as proposal than as a complete list of elementary types.

Also classification types refer to elementary values, but those are considered as separate type category.

### Categories

<b>any</b>	<i>Any kind of data</i> The type refers to any kind of data, where the current type of the data referenced is not known. It simply means that any type of data might be referenced.
<b>binary</b>	<i>Binary data</i> Binary is a generalized data type for any kind of non-textual and non-numeric data (e.g. movies, sounds, executables etc.).
<b>currency</b>	<i>Currency</i> Currency is used to express the value of a certain item. The type of currency (Euro, Dollar etc.) is not part of the conceptual type.
<b>date</b>	<i>Date</i> Date is considered conceptually as elementary value, although it usually consists of day, month and year.
<b>name</b>	<i>Name</i> The type name typically applies on characteristics, which consists of rather short text information used for identifying object instances in a collection. Usually, non-numeric identifying characteristics are associated with type name.
<b>number</b>	<i>Number</i> Numbers are used for quantitative characteristics in order to store the quantity of a certain item.

<b>text</b>	<p><i>Text</i></p> <p>Text type provides sufficient capacity for referring to text data of any size. Text is typically used as data type for descriptive characteristics.</p>
<b>time</b>	<p><i>Time</i></p> <p>Time is considered as elementary value from a conceptual point of view, although it usually consists of hour, minute, second etc. The precision of time type depends on the specific subject area, i.e. physicians may consider time values measured in nanoseconds or even more detailed, while the administration of a university is satisfied with time measured in hours and minutes.</p>
<b>timestamp</b>	<p><i>Timestamp</i></p> <p>Timestamps provide a time point including date and time or the number of seconds with any precision after a well-defined 0 time point. Similar to date and time, timestamp is considered as elementary from a conceptual point of view.</p>
<b>title</b>	<p><i>Title</i></p> <p>Title refers to text data with limited size (e.g. 200 characters). Similar to text, title is typically used as data type for descriptive characteristics.</p>
<b>yes/no</b>	<p><i>Yes or No</i></p> <p>Yes/No (Boolean) is used for properties, which are either true or false. Indeed, this is a classification, but conceptual, it is not considered as such and should be supported as elementary type in a terminology model.</p>

## 4.4.2 EventTypes - Event types

Event types are defined in order to provide appropriate event handling.

### Categories

#### transition

##### *State transition event*

State (transition) events are describing events as relevant state transitions, i.e. as state or state transitions that may cause a reaction, e.g. when a person has been born (cause) it has to be registered in the birth register (reaction).

#### temporal

##### *Temporal events*

Temporal events are describing events as consequence of time state transitions, e.g. when a person has birthday (cause). Temporal events may also include state transitions.

According to [UDT] state changes might be caused by value changes or changes in time. The question is, whether an object changes its state, just because time is passing by. Supposed, that an object has an *age*, the *age* will change with the time, and thus, the object will change its state. On the other hand is the fact that the object is getting older a consequence from changing time. Thus, it becomes more realistic, considering the changing time as base for the event rather than the object instance state reflected in *birth date*.

**process**

*Process events*

Process events are describing reactions by means of process event types, i.e. generic events that indicate changes in the process state of an object (being selected, being changed etc.). Process events may cause typical reactions in an application.

Process events are not caused by state changes in the sense of [UDT], but by changing the process state. Since process events are the events handled most frequently, those have been added to the terminology model, even though, they are of technical nature.

A list of typical process events follows:

**- read**

*After read event*

The event is generated after an object instance has been read.

**- store**

*Before store event*

The event is generated before storing an object instance.

**- modify**

*Before modify event*

The event is generated before modifying an object instance.

**- create**

*Before create event*

The event is generated before creating a new object instance.

**- remove**

*Before remove event*

The event is generated before removing an object instance.

**- delete**

*Before delete event*

The event is generated before deleting an object instance.

- insert**                      *Before insert event*  
The event is generated before inserting an object instance. The object instance does already exist.
- created**                      *After create event*  
The event is generated after a new object instance has been created.
- deleted**                      *After delete event*  
The event is generated after an object instance has been deleted.
- inserted**                      *After insert event*  
The event is generated after an object instance has been inserted.
- modified**                      *After modify event*  
The event is generated after an object instance has been updated.
- removed**                      *After remove event*  
The event is generated after an object instance has been removed.
- stored**                      *After store event*  
The event is generated after an object instance has been stored.

### 4.4.3 FeatureTypes - Feature types

Feature types define the object types supported as feature specializations. Features never appear as such, but as one of the listed specializations, only.

#### Categories

<b>rule</b>	<i>Rule feature</i> The feature is defined as rule → Rule
<b>event</b>	<i>Event feature</i> The feature is defined as event. → Event
<b>causality</b>	<i>Causality feature</i> The feature is defined as causality, i.e. it describes a reaction → Causality
<b>item</b>	<i>Item</i> The feature is an item and may provide several subordinated features. → Item
<b>- property</b>	<i>Property feature</i> The feature is defined as property. → Property
<b>-- association</b>	<i>Association</i> The property is an association. → Association
<b>-- part</b>	<i>Part</i> The property describes a part of an object. → Part
<b>-- generalization</b>	<i>Generalization</i> The property defines a generalization of an object type. → Generalization
<b>-- characteristic</b>	<i>Characteristic</i> The property is a characteristic. → Characteristic

- identifying**      *Identifying characteristic*  
An identifying characteristic is a characteristic, which is used for identifying instances in an object collection.
- quantifying**      *Quantifying characteristic*  
A quantifying characteristic is a characteristic, which provide quantities as object characteristic. Usually, quantifying characteristics refer to numeric type.
- classifying**      *Classifying characteristic*  
A classifying characteristic is a characteristic, which allows classifying object instances of a given type by categories. The type for classifying characteristics is a classification.
- descriptive**      *Descriptive attribute*  
A descriptive attribute is an attribute, which provides a textual description for an object.
- type**      *Object type*  
The feature is defined as type  
➔ Type
- elementary type**      *Elementary types*  
Elementary types are types (items) without properties. Elementary types may provide behavior and causalities.  
➔ ElementaryType
- classification**      *Classification*  
A classification defines a type that allows assigning individual objects to categories of the classification.  
➔ Classification
- category**      *Category feature*  
Category features are defined as categories of a classification or parent category. A category becomes a classification, again, when it refers provides a list of sub categories.  
➔ Category
- object type**      *Object type*  
The type (item) is defined as object type.  
➔ ObjectType

- key** *Key*  
A key is an object type defined as key. Keys are object types, which only contain characteristics.  
→ Key
- aggregation type** *Aggregation type*  
The object type is an aggregation type, which applies on classified collections.  
→ AggregationType
- terminology module** *Terminology module*  
The object type is defined as terminology module.  
→ TerminologyModule
- terminology model** *Terminology model*  
A terminology model is a terminology module.  
→ TerminologyModel

#### 4.4.4 ImportanceLevels - Importance levels

Importance levels describe the weight of concepts. The terminology model just provides a few categories, which might be extended when being used in an application.

##### Categories

- |                         |  |
|-------------------------|--|
| <b>problem relevant</b> | <i>Problem relevant concept</i><br>Problem relevant concepts are concepts, which are part of the subject area terminology, i.e. concepts that refer to terms used by subject matter experts. |
| <b>technical</b>        | <i>Technical concepts</i><br>Technical concepts are concepts that result from implementation issues. Usually, technical concepts are not of interest for subject matter experts.             |

## 4.4.5 RuleTypes - Rule types

Rule types define different purposes of rules. This list is not complete, but defines relevant rule types as known so far.

### Categories

<b>constraint</b>	<p><i>Constraint</i></p> <p>Constraints might be defined in order to decide, whether an object instance might become an element of the collection or not. On the other side, constraints might define validation rules in order to check the validity of properties.</p> <p>Constraints are rules, which return true in order to indicate that the instance or property is valid and false otherwise.</p>
<b>transition</b>	<p><i>State transition rule</i></p> <p>A state transition rule or action defines the way an object or related objects change under certain circumstances.</p>
<b>operation</b>	<p><i>Operation rules</i></p> <p>Operation rules create an additional property (result) referring to the derived concept. Operation rules may return elementary values, but also complex instances or collections. Typically, operation rules are defined as instance (row) operations, of collection operations. In general, however, operation rules may apply on any sort of arguments.</p>

## 5 References

- [TM] *Terminology Model*, RUN Software, Berlin, 2005  
[http://www.run-software.com/inhalt/downloads/documentation/P2\\_TerminologyModel\\_v1.pdf](http://www.run-software.com/inhalt/downloads/documentation/P2_TerminologyModel_v1.pdf)
- [TM1] Neuchâtel Group I: *Neuchâtel Terminology Model (Classifications)*, Stockholm, 2006  
[http://www.run-software.com/inhalt/downloads/P2c\\_TerminologyClassifications\\_v21.doc](http://www.run-software.com/inhalt/downloads/P2c_TerminologyClassifications_v21.doc)
- [TM2] Neuchâtel Group II: *Neuchâtel Terminology Model (Variables)*, Oslo, 2006  
[www.run-software.com/inhalt/downloads/documentation/P2b\\_TerminologyVariables\\_v1.doc](http://www.run-software.com/inhalt/downloads/documentation/P2b_TerminologyVariables_v1.doc)
- [1087] ISO 1087 Terminology work – Vocabulary  
ISO 704 Terminology work Principles - and methods
- [ODMG] R.G.G. Cattell, Douglas K. Barry: *The Object Data Standard: ODMG 3.0*  
Morgan Kauffmann Publishers 2003
- [UML] OMG Unified Modeling Language™  
<http://www.omg.org/spec/UML/2.4>
- [UDT] Karge R.: *Unified Database Theory*, RUN Software, Orlando (Florida), 7<sup>th</sup> World Multiconference on Systemics, Cybernetics and Informatics (SCI 2003)  
[http://www.run-software.com/inhalt/downloads/documentation/P1\\_UnifiedDatabaseTheory.pdf](http://www.run-software.com/inhalt/downloads/documentation/P1_UnifiedDatabaseTheory.pdf)