A technical discussion on modeling with UML
06/11/03

Rational. software


# Entity Relationship Modeling with UML

*Davor Gornik*

## *Table of Contents*

## *Entity Relationship Modeling*

One of the most misinterpreted terms in the software industry is actually one we know very well: entity relationship (ER). That's because we often lack a common definition that is understood by all members of the development team. We assume that everyone on the team shares the same clear understanding of the methodology, syntax, and mechanics associated with ER and ER modeling.

ER modeling itself defines the methodology used in the analysis and design of information-based systems. Database designers often use this methodology to gather requirements and define the architecture of the database systems. The output of this methodology is a list of entity types, relationship types, and constraints.

Unfortunately, ER modeling does not define the graphic syntax for the representation of ER diagrams. Many times notations are used solely by the database team and limit the ER modeling to relational database design. We need a notation that allows broader understanding by members of the entire system development team.

The Unified Modeling Language (UML) is a widely accepted language used by analysts and software developers that is an excellent fit for the graphic representation of ER diagrams. By using UML, development teams gain significant benefits, including easier communication between team members, easy integration to repositories due to this language based on meta-models, use of a standardized input/output format (XMI), universal use for application and data modeling, unified representation from analysis to implementation to deployment, and completeness of specification.

This white paper defines the core concepts of ER modeling and explains how UML can be used by development teams to develop ER models.

## *Core Elements of ER Modeling*

ER modeling is based on artifacts, which can be either a representation of physical artifacts, such as Product or Employee, or a representation of a transaction between artifacts, such as Order or Delivery. Each artifact contains information about itself. ER modeling also focuses on relationships between artifacts. These relationships can be either binary, connecting two artifacts, or ternary, among several artifacts.

The four essential elements of ER modeling are:

- Entity types
- Attributes
- Relationship types
- Attributes on relationships

### Entity Types

An entity type is a set of artifacts with the same structure and independent existence within the enterprise. Examples of an entity type would be Employees or Products.

A single occurrence of an artifact is an entity. While an entity type describes the structure, the entity itself identifies the single instance and all of the data of this instance. An example of an Employees entity would be the Employee Joe Ward.

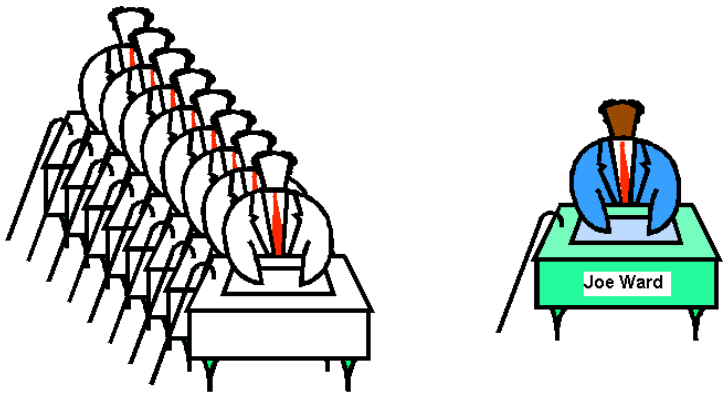


Figure 1 Entity Type Employees and Entity Employee Joe Ward

## Attributes

The structure of an entity type is defined with attributes. An attribute can be seen as a property of an entity type. Attributes of an Employee might be Name, Address, Social Security Number, Birth Date, Date Joined, and Position.

Entities differ from each other by the values of their attributes. Since it is possible to have entities with exactly the same values for attributes, we lose the ability to differentiate and address a specific entity. Therefore, we must always make sure that the values of attributes of a specific entity are unique to values of other entities. Each Employee has a unique combination of Name and Social Security Number attributes.

An example of associated values for attributes of an Employee is: Joe Ward, living at 34 Main Road, Redmond, WA, 98053, has the Social Security Number 555-32-2222, was born on September 7, 1971, and joined our company October 1, 2001, as a service engineer for consumer electronics.

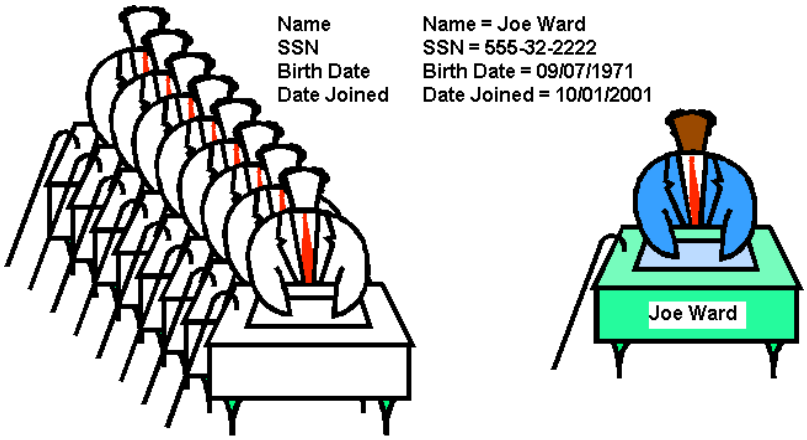


Figure 2 Attributes of the entity types Employees and the values for attributes for the entity Employee Joe Ward

## Relationship Types

While entity types describe independent artifacts, relationship types describe meaningful associations between entity types. To be precise, the relationship type describes that entities of entity types participating in the relationship can build a meaningful association. The actual occurrence of the association between entities is called a relationship.

It is important to understand that although we defined a relationship type, this does not mean that every pair of entities builds a relationship. A relationship type defines only that relationships can occur.

An example of a relationship type is the Employee owns Product. In this example the relationship type is Owns. The relationship itself is: The Employee Joe Ward owns a Product, the yellow Telephone, with the Serial Number 320TS03880.

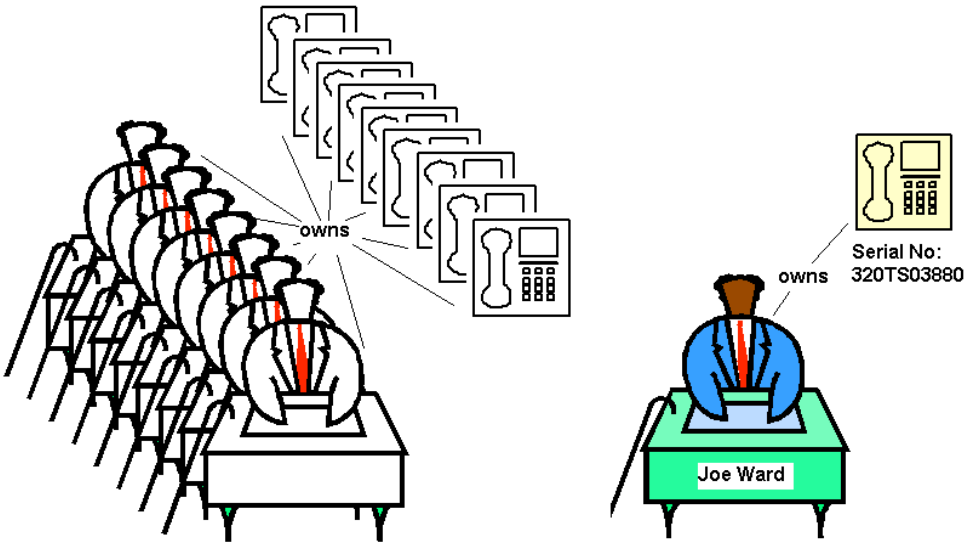


**Figure 3 Relationship Type owns and Relationship owns between Employee Joe Ward and the Product with the serial number 320TS03880**

There might be a second employee Martin Weber who does not own a Telephone.

### Attributes of Relationship Types

Relationship types can also contain attributes. For example the relationship type Services between the Employee and the Product could contain the attributes Date and Status, which identify the date of the service and the status of the product after the service is done.

When the relationship is realized in a concrete occurrence of the service, the values of the attributes of the relationship are set. The meaning of the relationship could be: Joe Ward services the black Toaster with the Serial Number 0462834DF4 on July 3, 2002, and establishes the status as good working condition.

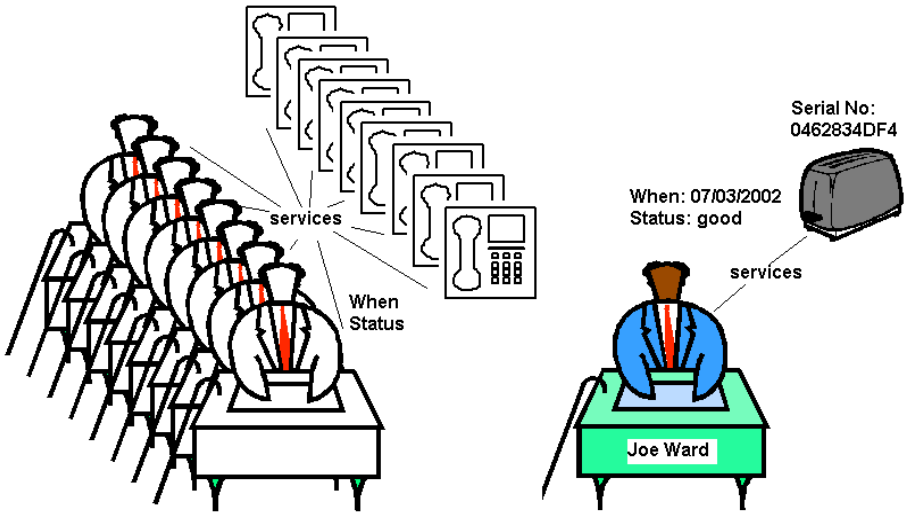


**Figure 4 Attributes on Relationship Type services and Attributes on Relationship Joe Ward services Product with the Serial No 0462834DF4**

## Simple Constraints in ER Modeling

Entities, relationships, and attributes within the ER model establish restrictions that define the structure of the enterprise. The structure is limited by rules called constraints. For example, it is not feasible that an Employee deals with more than 100 customers. Or, every employee must be associated with exactly one department.

### *Cardinality*

Each specified relationship type defines the possibility of establishing relationships between all of the participating entities. In most cases, this is not necessary. For example, not all Employees own all of the Products.

Relationships are bi-directional, connecting two entity types (Employees and Products) or the same entity type playing two different roles (Employees as a manager and Employees as a subordinate). Relationships can also be multi-directional, connecting more than two entity types. One example of a multi-directional relationship would be a phone call connecting an employee, a customer, and two phones. In either case, each entity type specifies the cardinality towards the relationship type.

The simplest cardinality is specified by the number of relationships allowed per entity. If only one Department participates in the relationship associated with an Employee, we write a 1 on the connector. This would mean that Joe Ward must be associated with one and only one Department.

Other possibilities for cardinality are either Not Specified or Specified By a Variable. The Not Specified cardinality is unlimited. The same is true for a cardinality Specified By a Variable (mostly M or N).

When the lower and upper limits of participating entities in a relationship are different, we specify a pair of values for the lower and higher limits enclosed in parenthesis and separated by a comma (M, N). An optional relationship would be recognized by (0, 1) or (0, N) dependent on the upper limit.

For example, a setup for Players on a Soccer team would be something like (11, 18). The entity Redmond Lions Soccer Team builds a relationship to the entity type Players, which consists of Joe Coplen, David Archer, John Good, Kevin Hale, Ivan Komashinsky, Steven Cooper, Andrew Bliven, Art Lounsbery, Chad Beery, Randall DuBois, Ron Baghai, Lance Delo, Tito Magobet, Curtis Hrischuk, and Ian Leslie.

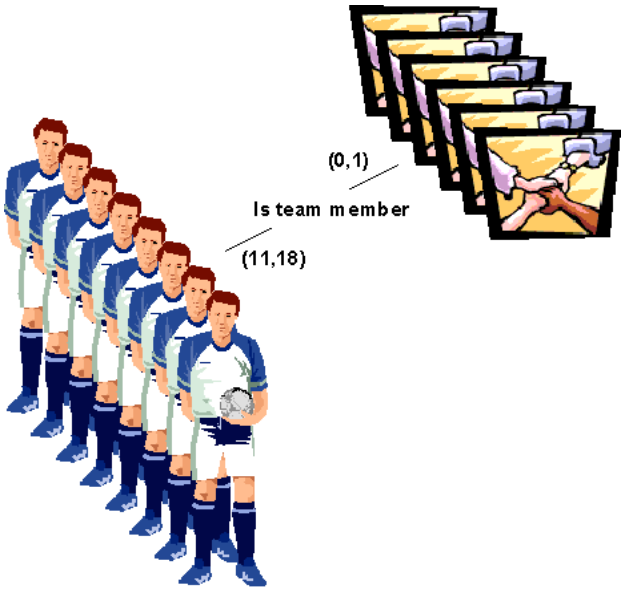


Figure 5 The Relationship between Soccer Teams and Players defines that the relationship is valid only when the Soccer Teams type relates to 11 to 18 Players

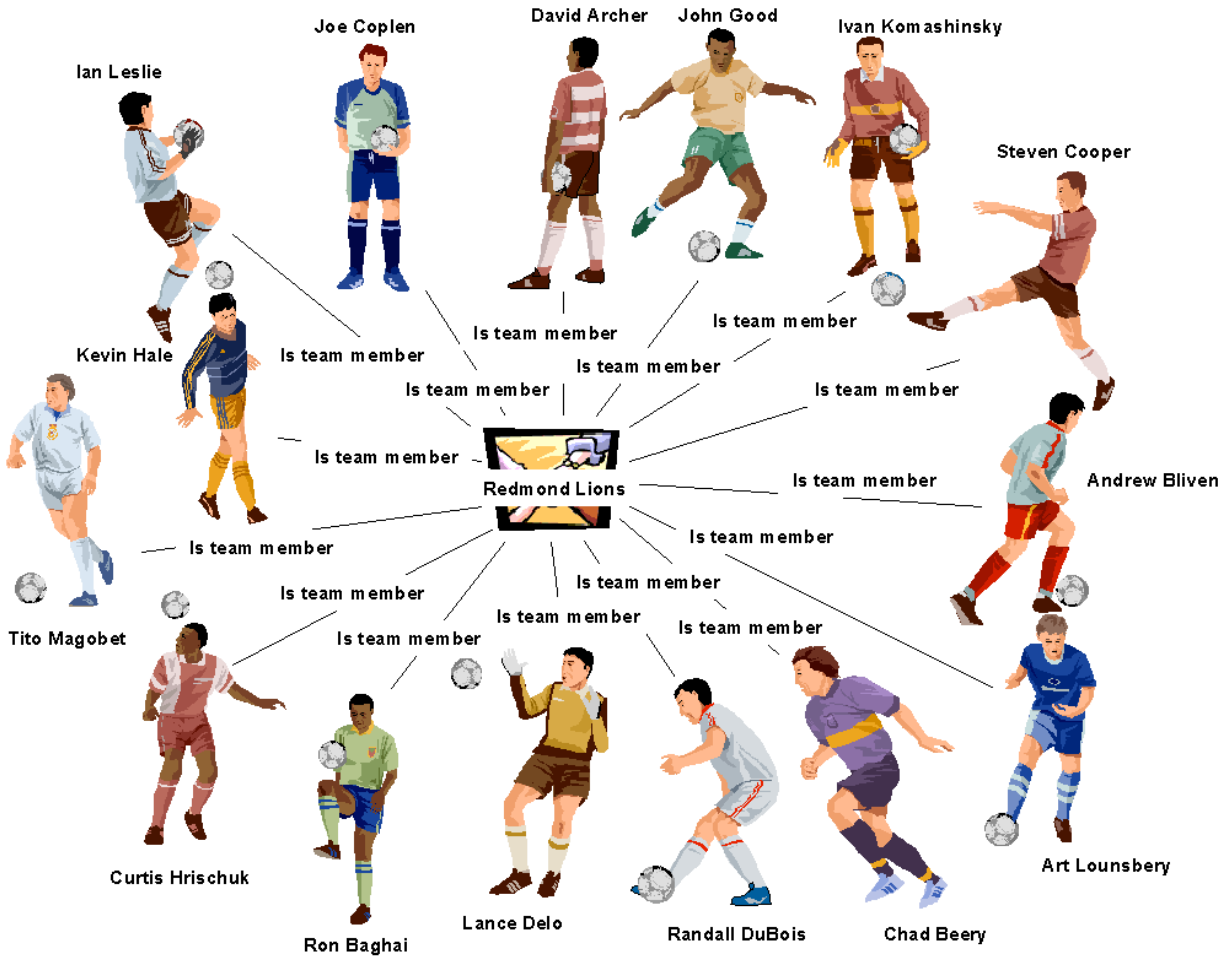


Figure 6 A valid relationship is team member between the Redmond Lions soccer club and 15 Players

There are several occasions when soccer players receive yellow or red cards for bad behavior or fouls. They are described in the Cards entity type. The Cards entity type builds a Received relationship type with Players with a cardinality of (0, N). This means that the Player David Archer can have a Received relationship to 3 Card entities, whereas Lance Delo does not have any.

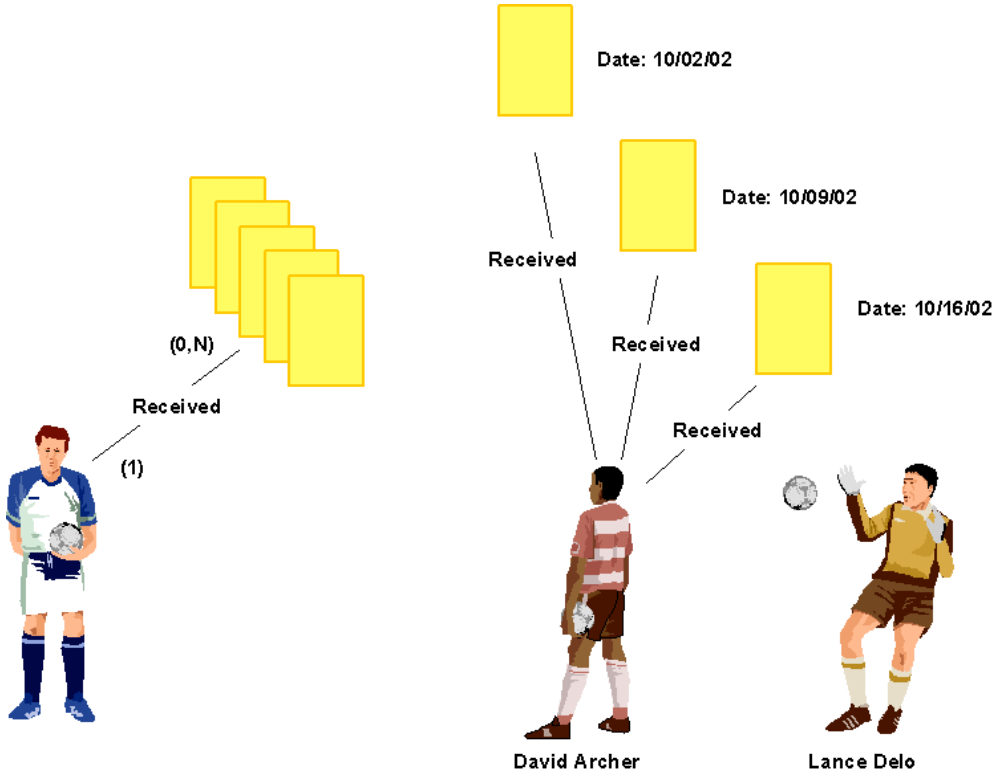


**Figure 7 Entity type Players can Receive 0 or any number of entity type Cards: player David Archer received 3 cards, player Lance Delo did not receive any cards yet**

### *Dependency*

Dependency refers to a situation in which a particular entity can not meaningfully exist without the existence of another entity as specified within a relationship. The entity type is dependent on another entity type when each entity of a dependent entity (subtype) depends on the existence of the corresponding parent entity in the super type.

A mandatory dependency relationship has to be specified by explicitly defining the lower limit for cardinality that is not equal to 0 or by a fixed cardinality that is not equal to 0.

An example of a dependency between two entities would be the entity type Marriage Certificate, which depends on the entity type Person. The relationship is Married with a fixed dependency on one Marriage Certificate and two Persons.

For example, the entity Marriage Certificate 352647003 has a fixed dependency to entities Joe Ward and Melinda Bell. This means that if either Melinda Bell or Joe Ward leaves the relationship, the Marriage Certificate 352647003 becomes invalid – at least from the data point of view.

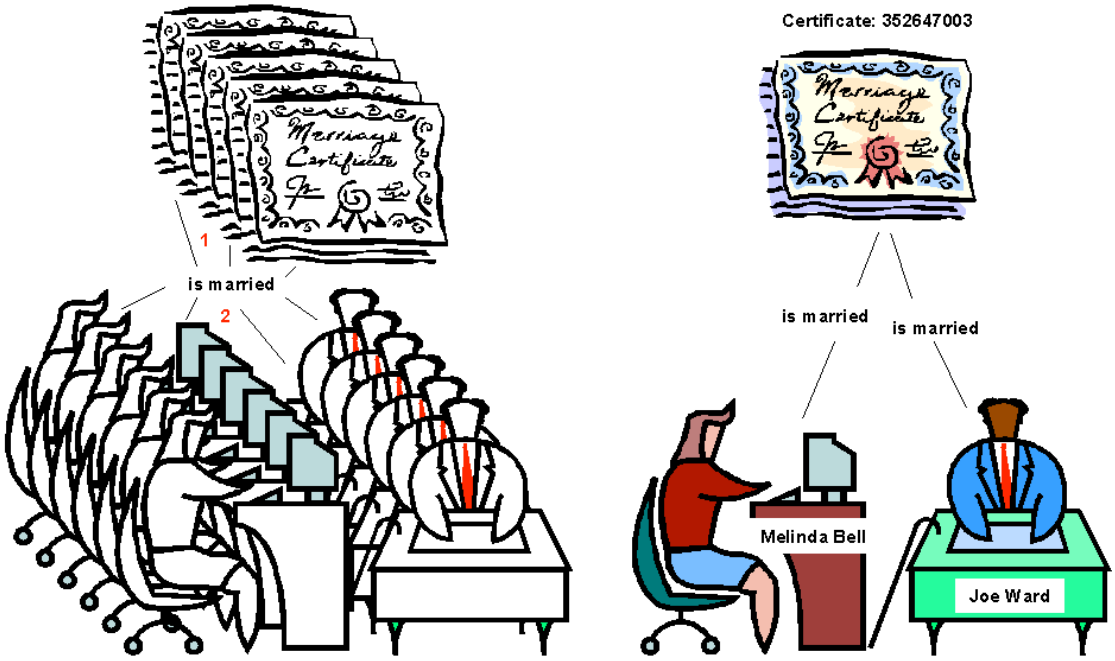


**Figure 8 Dependent entity type Marriage Certificate on the entity type Persons and dependency of the entity Marriage Certificate 352647003 on Melinda Bell and Joe Ward**

## Specialization and Generalization

The core ER model defines only basic relationships between entity types. While the basic entities and relationships can easily represent most of the simple data structures in business organizations, technical applications require more complex structures based on similarity and differences between entity types.

### *Specialization and Generalization*

The intent of specialization and generalization is reuse of the attributes and behaviors associated with entity types.

Specialization is used to define an entity type that represents a specific segment of a larger entity type. The specialized entity type inherits the structure and behavior, such as business rules, from the parent entity type. However, while the specialized entity type extends the parent structure or behavior, it is never less than the parent.

For example, Employees is a specialization of the entity type Persons, which requires all of the attributes and relationships of the entity type Persons. There may also be a second entity type called Customers that is a specialization of the entity type Persons. Both entity types have the same Persons attributes, which are seen as attributes of Employees or attributes of Customers. Therefore, when looking at the Customers, we see all of the attributes specified in the Persons entity type and specified in the Customers entity type.

Generalization is exactly the opposite workflow. The generalized entity type (or parent type) represents the common structure and behavior of all subtypes and contains all of the common attributes from child entity types. The child entity types have a complete view of the parent attributes and their own attributes.

The process of generalization finds the common structure and extracts it in the parent entity type. The parent entity type is commonly found during refactoring when comparing entity types and simplifying the model.

While specialization can be directly implemented only with object-oriented or object-relational databases, generalization can be directly implemented with any relational database using foreign keys.

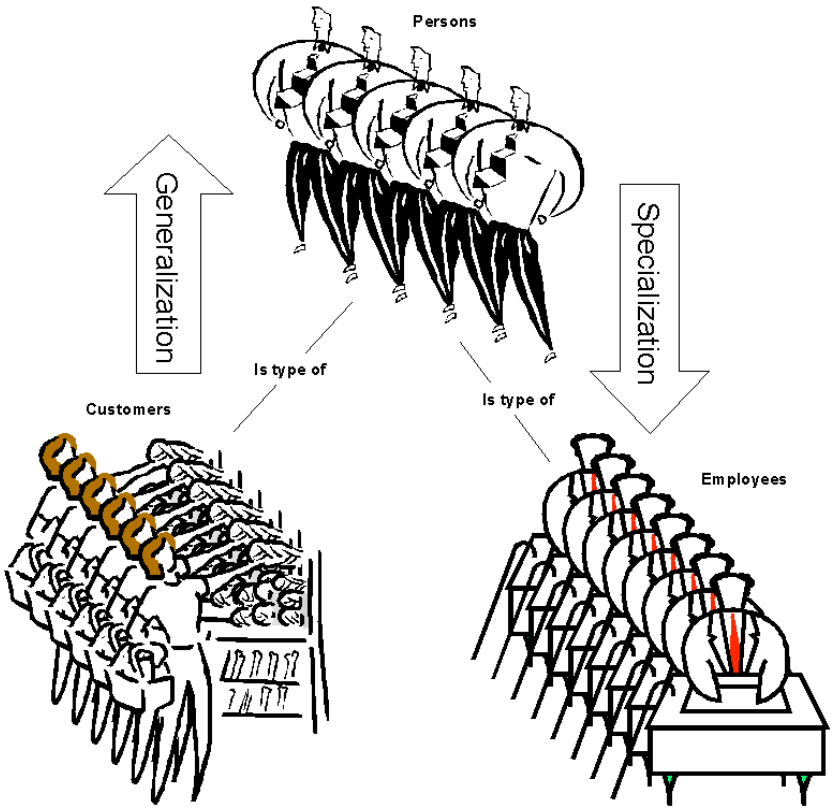


**Figure 9 Generalization of Customers entity type and Employees entity type to Persons entity type; Specialization works in the opposite direction**

### Categorization

While specialization is done on entities, categorization defines constraints on relationship types. In most cases, categorization is exclusive, meaning that one entity participates in either relationship A or relationship B, depending on the entities' status. The status can be either a value of an attribute, a presence of another relationship, or some external status.

Categorization does not change the attributes of an entity. It requires the data access and manipulation to consider the constraints specified in the categorization.

A good example of categorization is Vehicles. Depending on the kind of vehicle, we need to build different relationships. For trucks, we need the cargo information, while for busses, we need the names of passengers. This information will be used in different relationships to deliver meaningful context to the relationship.

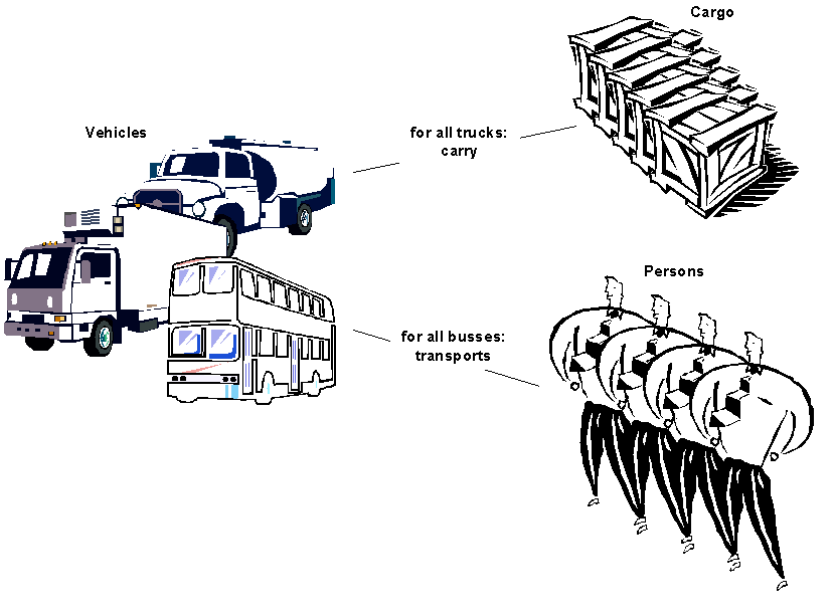


**Figure 10 Categorization in trucks and busses with relationship types to Cargo entity type and Persons entity type depending on the category**

## *Notations for the ER Methodology*

Today, several notations are used within ER modeling, including Chen's ER, Barker ER Information Engineering (IE), and IDEF1X, most of which also cover the relational notation.

Object role modeling (ORM) notation is also a player in the ER methodology. It is able to express very precise and complete business rules and can illustrate constraints graphically. Unfortunately, this level of detail requires huge diagrams with plenty of details. The question related to ORM is whether we need the precision of the notation at the level of the ER model. The ORM notation is also very complex and differentiates completely from other notations, making it less understandable to project team members not using ORM.

UML is a notation language initially created for software design that has expanded into business and database design. It includes elements and diagrams necessary to specify everything from analysis to implementation to deployment. By employing several types of diagrams and tens of different elements, UML is able to express different levels of abstraction for a system. This is a very unique capability. However, we do not need to know all of UML or every possible view and representation to successfully use UML for ER modeling.

To better understand the role that UML plays in the ER methodology, we will describe how UML handles each of the core elements of ER modeling that have been outlined earlier in this white paper.

### Entity Types in UML

As was mentioned earlier, an entity type identifies a series of artifacts of the same structure. The entity type is a blueprint that can produce any number of artifacts that differ from each other only by their identity and status.

The corresponding element of the UML is the class. The class by definition has the ability to hide the content, while the entity has accessible interfaces. This seems to be contradictory, but it is not. UML allows the class to publicize the structure using public attributes.

Classes are drawn as rectangles with up to three compartments:

- Compartment one includes the stereotype and the name of the class. Stereotype refers to the further classification of elements in UML to enforce common characteristics. For example all of the legacy classes could carry the stereotype legacy to immediately classify them as not modifiable. While the class itself is a representation of a type, we classify the type with the stereotype <<Entity>> (<<…>> is the syntax used to specify the stereotype).

- Compartment two contains attributes with types and visibility. It can also contain other details about attributes such as initialization value and stereotype. The second compartment can be omitted when displayed in overview diagrams.
- Compartment three is reserved for the behavior of the class. Since the entity type does not need the behavior, we will omit this compartment.

The class can be displayed with one, two, or three compartments on diagrams depending on the level of abstraction.

An entity is an instance of the entity type. In UML, object is an instance of a class. This means that the entity itself corresponds to the object.

The representation of the object is derived from the representation of the class. The most visible difference is that the name of the object is underlined and that there are only one or two compartments.

The first compartment contains the optional stereotype, the name of the object, and the name of the derived class, divided by a colon. At least one of the names must be specified. The second compartment contains the relevant attributes with their values.

A great way to represent the object is to use just one compartment and specify the identifier[1] as the name of the object.

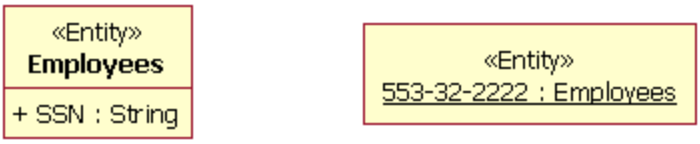


**Figure 11 Entity type Employees and entity 553-32-2222 displayed as class and object in UML**

### Attributes in UML

The status and information about the entity is stored as attributes of an object. The attributes of entity types have to be visible – or public – to other entity types. The attributes are specified with the visibility, name, and type in the class specification. The type of the attributes is the analysis type. It may change during the design phase.

The attributes are specified in the second compartment of the class. They contain the visibility specification that is always "+" (public) for entities. The name is divided from the type by a colon.

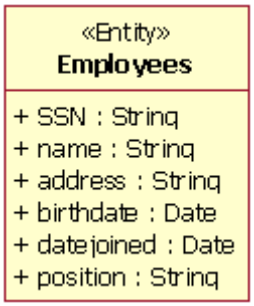


**Figure 12 Entity Type Employees with public attributes SSN, name, address, birth date, date joined, and position**

### Relationship Types in UML

All relationships between classes in UML are specified for the type and not for the instance. The numbers at both ends of the relationship specify the cardinality: the number of possible instances participating in the relationship.

The name of a relationship is specified directly on the relationship line and is used to identify the relationship. It can help a reader understand why the relationship exists. If a relationship is not named, role names may be used to help a reader

[1] Identifier is the chosen attribute that uniquely identifies an object from other objects.

understand a relationship. The figure below can be read as Products are owned by Employees or Employees are owner of Products. The singular and plural wording in the description of the relationship is defined by cardinality.

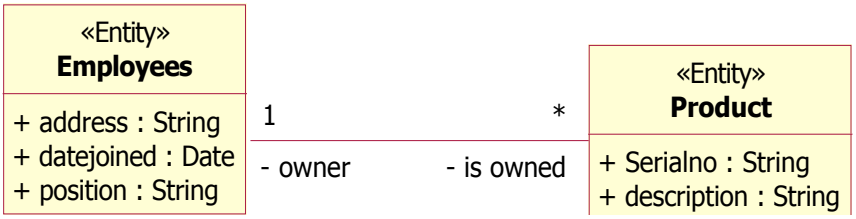


**Figure 13 Relationship between the entity types Employee and Product**

The data types used in the entity types are not standardized. Users are allowed to use any data type needed. It is a good practice to create a glossary with all data types to allow standardization and understanding across several designers and projects within a company.

### Attributes of Relationship Types in UML

A relationship can have attributes that are shown in UML association classes. The association class is displayed in a rectangle and contains the list of public attributes for the relationship. The association class is attached to the relationship with a dotted line. The class does not need a stereotype to explain the use and classify the class because the attachment already defines it.

The attributes in the association class are specified with the visibility, name, and type.

Although it seems like the association class, the attachment, and the relationship are independent elements, they actually represent the same element. The names of the relationship and the association class must be related.

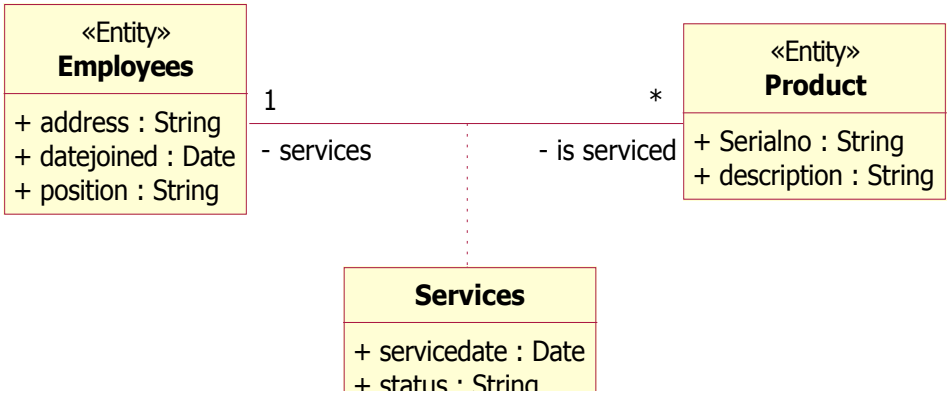


**Figure 14 Attributes of a relationship type Services are specified in the association class**

### Simple Constraints in UML

#### *Cardinality*

UML defines a very consistent way to specify cardinality. It is always specified by numbers at each end of relationships. Possible definitions include a single number for a specified cardinality of certain number of instances, which could be also unlimited, and a pair of numbers divided by ".." that specify a range for cardinality. The symbol used for unlimited cardinality is "*" and can be used either alone, meaning an optional unlimited relationship, or in combination with another low value to specify the mandatory relationship (like "1..*"). The values for lower and upper limits of the cardinality can be any positive number or "*", where the first number must be smaller or equal to the second one.


**Figure 15 The SoccerTeam entity type defines a relationship plays to 11 to 18 players of the entity type Player**

### *Dependency*

UML distinguishes two forms of dependencies between entity types. An aggregation is a dependency between two entity types that is required for the existence of the dependent entity type. The syntax for aggregation in UML is a hollow diamond on the side of the aggregate. The same side has a mandatory cardinality of 1, which can be omitted.

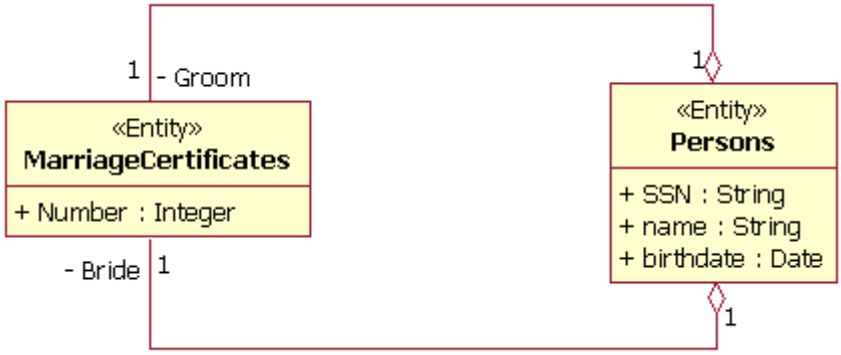


**Figure 16 Each MerriageCertificates entity type depends on two Persons in the role of a Bride and a Groom**

Aggregation is used when the aggregate is not unique for all of the dependencies and not all of the dependent instances must relate to the same entity. In case the aggregate is a single entity for all of the dependencies, UML specifies a strong dependency called composition. Composition is represented in the graphics as a filled diamond on the aggregate side. This relationship is used when the aggregate contains the subordinate entity type.


**Figure 17 The entity type OrderPositions is completely defined by the entity type Orders as specified by the composition**

Cardinality can be combined with aggregation and composition to define constraints on these relationships.

## Specialization and Generalization

An essential part of analysis is dedicated to the similarities and differences of entity types. Specialization reduces the risks of under specification and reduces the lack of requirements by inheriting requirements from the parent. Generalization simplifies the model and the implementation of entity types in the system.

Generalization is defined in UML by a hollow arrow on the parent side of the relationship. Generalization is not a relationship between two entity types. It is a derivation of the general entity type to the specialized entity type. Cardinality is not allowed on generalization relationships.

All of the attributes and relationships of the parent entity type are inherited by the specialized entity type. Both the attributes and relationships to other entity types cannot be removed from the specialized entity type.

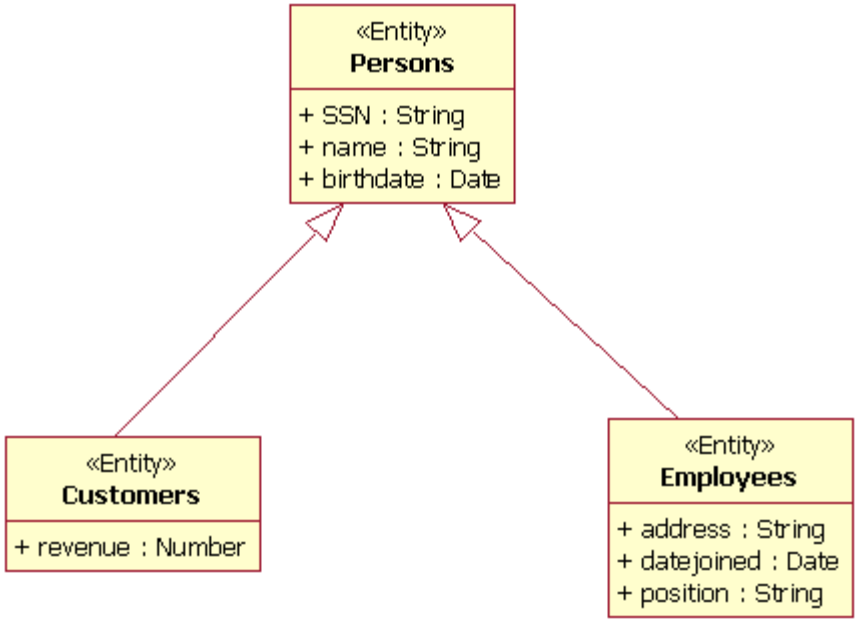


**Figure 18 Generalization defines the entity types Employees and Customers as type of the entity type Persons inheriting Persons attributes**

### Categorization

Categories of the same entity type specify how the entities relate to each other depending on their characteristics, like all employees on leave of absence for example. The syntax of the categorization uses the constraints specified in OCL (object constraint language) connected to the relationship.

The constraints are meant to specify dynamic behavior. When constraints evaluate as valid, the relationships are valid.

Typically the constraints are mutually exclusive, which can be modeled using a constraint {xor} between exclusive relationships.

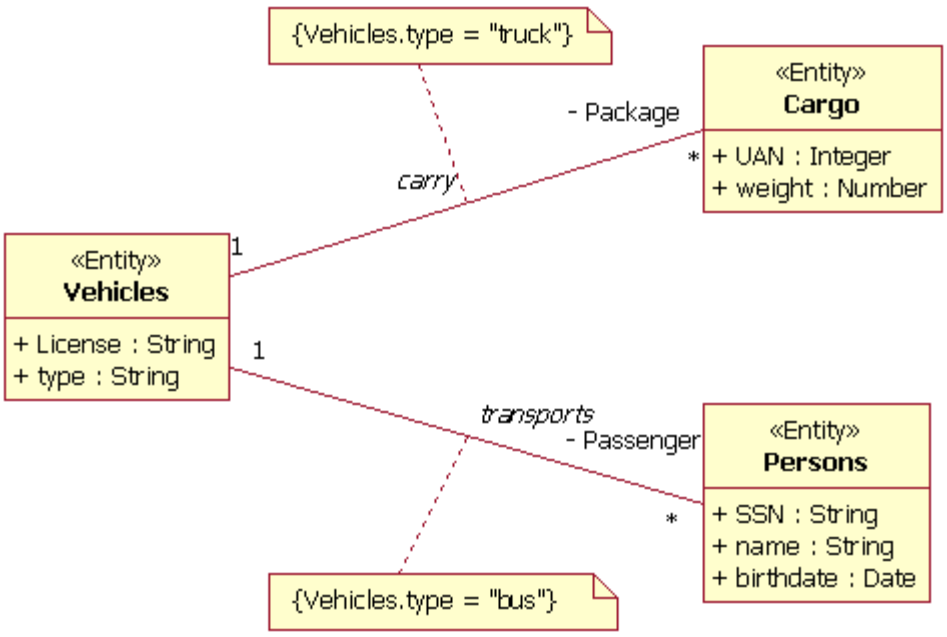


**Figure 19 Categorization of entity types defines the criteria for relationship types – Cargo is important for Vehicles of the type "truck"; Persons can be transported for the Vehicles of the type "bus"**

## *Summary*

Against the common opinion, ER methodology is not limited to development of relational databases. I have to agree that in most cases the output of the design process will be realized in a relational database; however this is not a condition for it. ER methodology is focused on artifact-based design. Its output is artifacts (entity types) with relationships between them and constraints clarifying entity types and relationships. This output can be used to create a relational model that will have additional technology-related constraints.

The ER methodology is used for analysis and design of artifact-driven systems. It is used for conceptual and logical modeling, but is not intended for physical design. The ER methodology describes only the static view of the artifacts, not the dynamics of the system.

To create a smooth development process, it is important that all members of the development team "speak a common language." That's because misinterpretation of information can cause delays, create unexpected errors, and reduce the overall efficiencies of team members. UML helps eliminate these concerns by providing a standardized language that is easily understood by all members of the system development team.

**IBM software integrated solutions**

IBM Rational supports a wealth of other offerings from IBM software. IBM software solutions can give you the power to achieve your priority business and IT goals.

- *DB2® software helps you leverage information with solutions for data enablement, data management, and data distribution.*
- *Lotus® software helps your staff be productive with solutions for authoring, managing, communicating, and sharing knowledge.*
- *Tivoli® software helps you manage the technology that runs your e-business infrastructure.*
- *WebSphere® software helps you extend your existing business-critical processes to the Web.*
- *Rational® software helps you improve your software development capability with tools, services, and best practices.*

**Rational software from IBM**

Rational software from IBM helps organizations create business value by improving their software development capability. The Rational software development platform integrates software engineering best practices, tools, and services. With it, organizations thrive in an on demand world by being more responsive, resilient, and focused. Rational's standards-based, cross-platform solution helps software development teams create and extend business applications, embedded systems and software products. Ninety-eight of the Fortune 100 rely on Rational tools to build better software, faster. Additional information is available at www.rational.com and www.therationaledge.com, the monthly e-zine for the Rational community.



IBM Corporation
Software Group
Route 100
Somers, NY 10589
U.S.A.

Printed in the United States of America 05-03 All Rights Reserved. Made in the U.S.A.



The IBM home page on the Internet can be found at **ibm.com**

TP202