利用 UML 进行实体关系建模

软件行业中最常被误解的一个术语实际上是我们非常熟悉的一个：实体关系（ER）。这是因为我们经常缺少一种能被开发团队的所有成员理解的共同定义。我们假定团队的每个成员都对与 ER 和 ER 建模相关的方法学、语法和机制（mechanics）有着同样清楚的理解。

Davor Gornik, 高级工程师

2004 年 11 月 01 日

内容

实体关系建模

软件行业中最常被误解的一个术语实际上是我们非常熟悉的一个：实体关系（ER）。这是因为我们经常缺少一种能被开发团队的所有成员理解的共同定义。我们假定团队的每个成员都对与 ER 和 ER 建模相关的方法学、语法和机制（mechanics）有着同样清楚的理解。

ER 建模本身定义了在基于信息的系统的分析和设计中用到的方法。数据库设计者通常使用该方法来收集需求，并定义数据库系统的构架。该方法的输出是实体类型、关系类型和约束条件的清单。

不幸的是，ER 建模没有为 ER 图的表示定义图解语法。数据库团队经常单独使用表示法，并且将 ER 建模限制在关系数据库设计的范围内。我们需要一种能让整个系统开发团队的成员获得更广泛理解的表示法。

统一建模语言（UML）是一种分析人员和软件开发人员广泛使用的语言，特别适合 ER 图的图形化表示。通过使用 UML，开发团队受益匪浅，这些获益包括团队成员间的交流更加简单，由于该语言是基于元模型的因而更容易与知识库集成，标准化输入/输出格式（XMI）的使用，应用建模和数据建模的普遍使用，从分析到实施再到部署的统一表示，以及规格说明书的完整性。

本白皮书定义了 ER 建模的核心概念，并解释了开发团队如何能够利用 UML 开发 ER 模型。

ER 建模的核心要素

ER 建模基于工件，可以是物理工件（比如 Product 或 Employee）的表示或者工件（比如 Order 或 Delivery）之间事务的表示。每个工件都包含关于自身的信息。ER 建模还专注于工件间的关系。这些关系可以是二元的（连接两个工件），也可以是三元的（在几个工件之间）。

ER 建模的四个必要元素是：

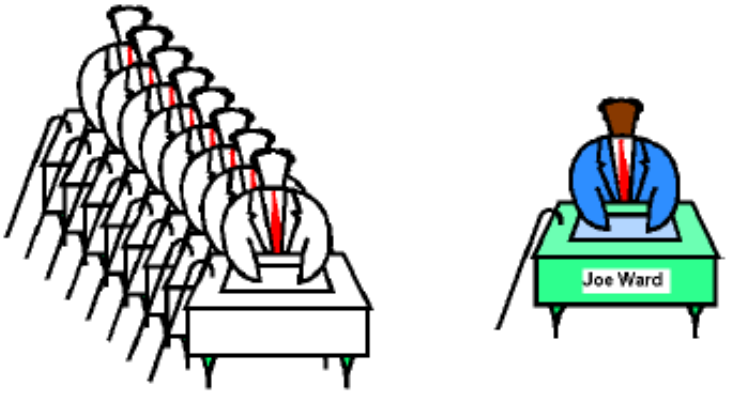
1. 实体类型
2. 属性
3. 关系类型
4. 关系属性

实体类型

实体类型是具有相同的结构并在企业内部独立存在的一组工件。Employees 或 Products 就是实体类型的例子。

工件的一次出现就是一个实体。虽然实体类型描述了结构，但是实体本身标识了单个实例以及该实例的所有数据。Employee Joe Ward 就是 Employees 实体的一个例子。

图 1 实体类型 Employees 和实体 Employees Joe Ward



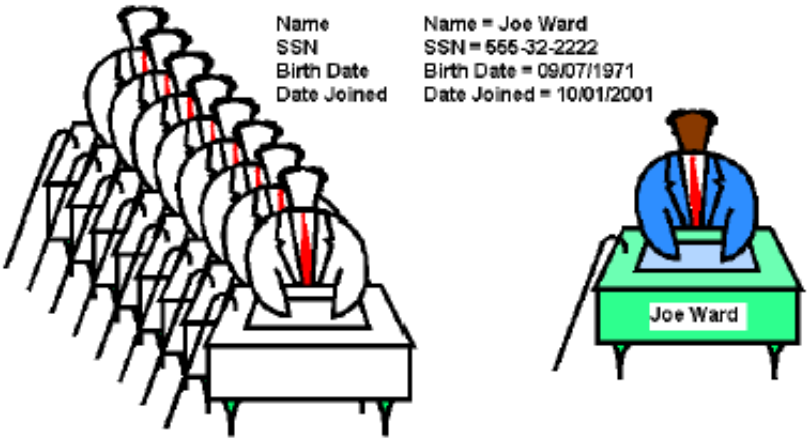
属性

实体类型的结构是用属性定义的。属性可看作实体类型的特征。Employee 的属性可能有姓名、住址、社会安全号码、出生日期、参加工作日期以及职务等。

实体通过属性值来互相区分。由于实体的属性可能有相同的值，如果这样我们就不能区分某些特定的实体。因此，我们必须保证特定实体的属性值与其他实体的属性值不同。各个 Employee 都有一个唯一的姓名和社会安全号码属性组合。

Employee 的属性值的一个例子是：Joe Ward，地址为 34 Main Road, Redmond, WA, 98053，社会安全号码为 555-32-2222，出生于 1971 年 9 月 7 日，2001 年十月 1 日加入公司，是家电服务工程师。

图 2 实体类型 Employee 的属性以及实体 Employee Joe Ward 的属性值



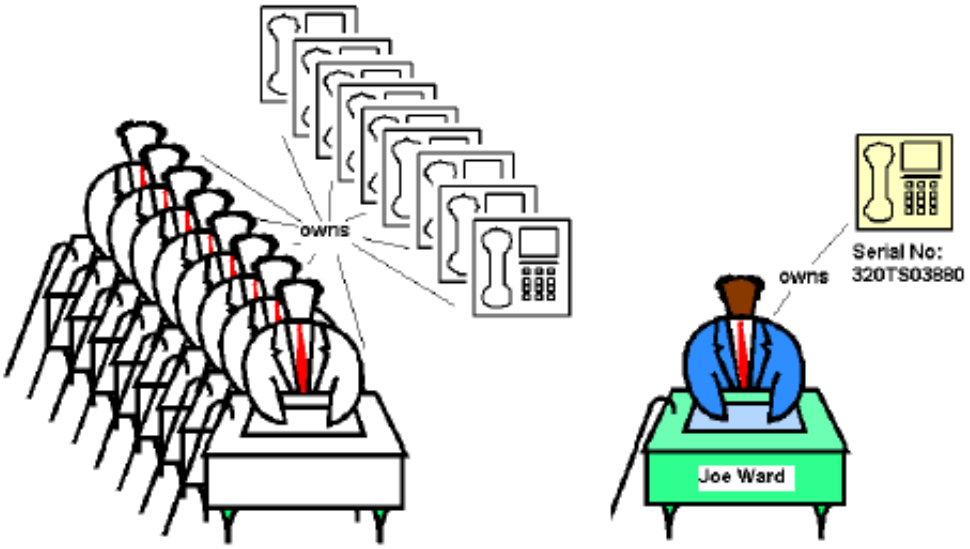
关系类型

实体类型描述了独立工件，而关系类型描述了实体类型间有意义的关联。更准确的说，关系类型描述了参与该关系的实体类型的实体可以构建一个有意义的关联。实体间实际发生的关联被称为一种关系。

有一点我们必须理解：尽管我们已经定义了一种关系类型，但是这并不意味着每一对实体都构建了一种关系。关系类型规定了发生关系的类型。

关系类型的一个例子是 Employee 拥有 Product。在该例中，关系类型是 Owns。关系本身是：Employee Joe Ward 拥有一种产品--一部编号 320 TS 03880 的黄色电话机。

图 3 雇员 Joe Ward 和序号 320 TS 03880 的产品之间的拥有关系类型和拥有关系



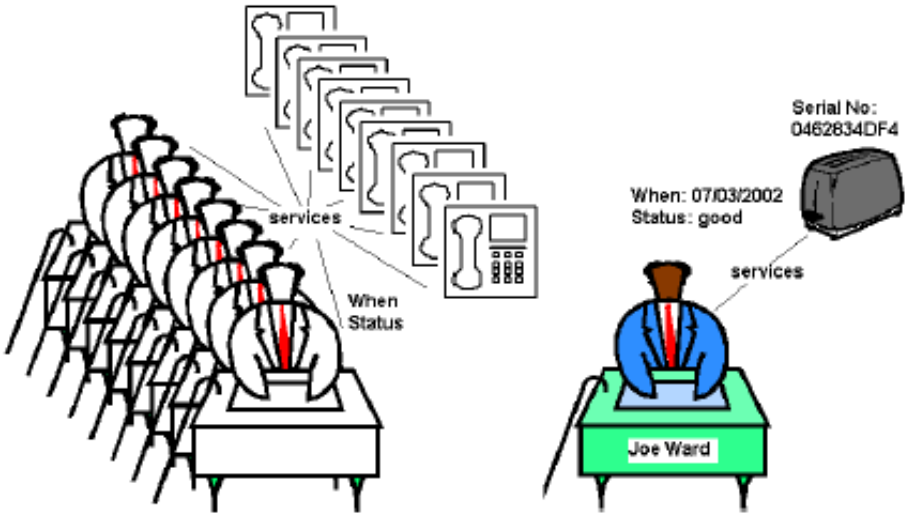
也有可能有一位名叫 Martin Weber 的雇员就没有拥有电话机。

关系类型的属性

关系类型还可以包含属性。比如，Employee 和 Product 间的关系类型 Services 可以包含属性 Date 和 Status，标识了服务的日期以及服务之后产品的状态。

当在具体发生的服务中实现关系时，该关系的属性值就被设置。关系的含义可能是：Joe Ward 在 2002 年 7 月 3 日为序号是 0462834 DF 4的黑色饮水机提供维修服务，并且使其建立了良好的工作状态。

图 4 services 关系类型的属性，以及 Joe Ward 为序号 0462834 DF 4的产品提供服务的关系的属性



ER 建模中的简单约束

ER 模型中的实体、关系和属性建立了一些定义企业结构的限制。该结构受被称为"约束"的规则所限制。比如，一个雇员不可能与 100 多位客户打交道。或者说，每个雇员必须与某一个恰当的部门关联。

基数（Cardinality）
每个指定的关系类型都定义了在所有参与实体之间建立关系的可能性。大多数情况下，这不是必需的。比如，并非所有 Employee 都拥有全部 Product。

关系是双向的，连接了两种实体类型（Employees 和 Products）或者扮演两个不同角色（作为经理的 Employee 和作为下属的 Employee）的同一实体类型。关系还可以是多向的，连接两个以上的实体类型。连接一个雇员、一个客户和两部话机的一次电话呼叫就是多向关系的一个例子。不管是哪种情况，每个实体类型都指定了针对该关系类型的基数。

通过每个实体最大的关系数目指定了最简单的基数。如果只有一个部门参与和一个雇员关联的关系，那么我们在连接器上写一个 1。这意味着 Joe Ward 必须与一个并且只能是一个部门关联。

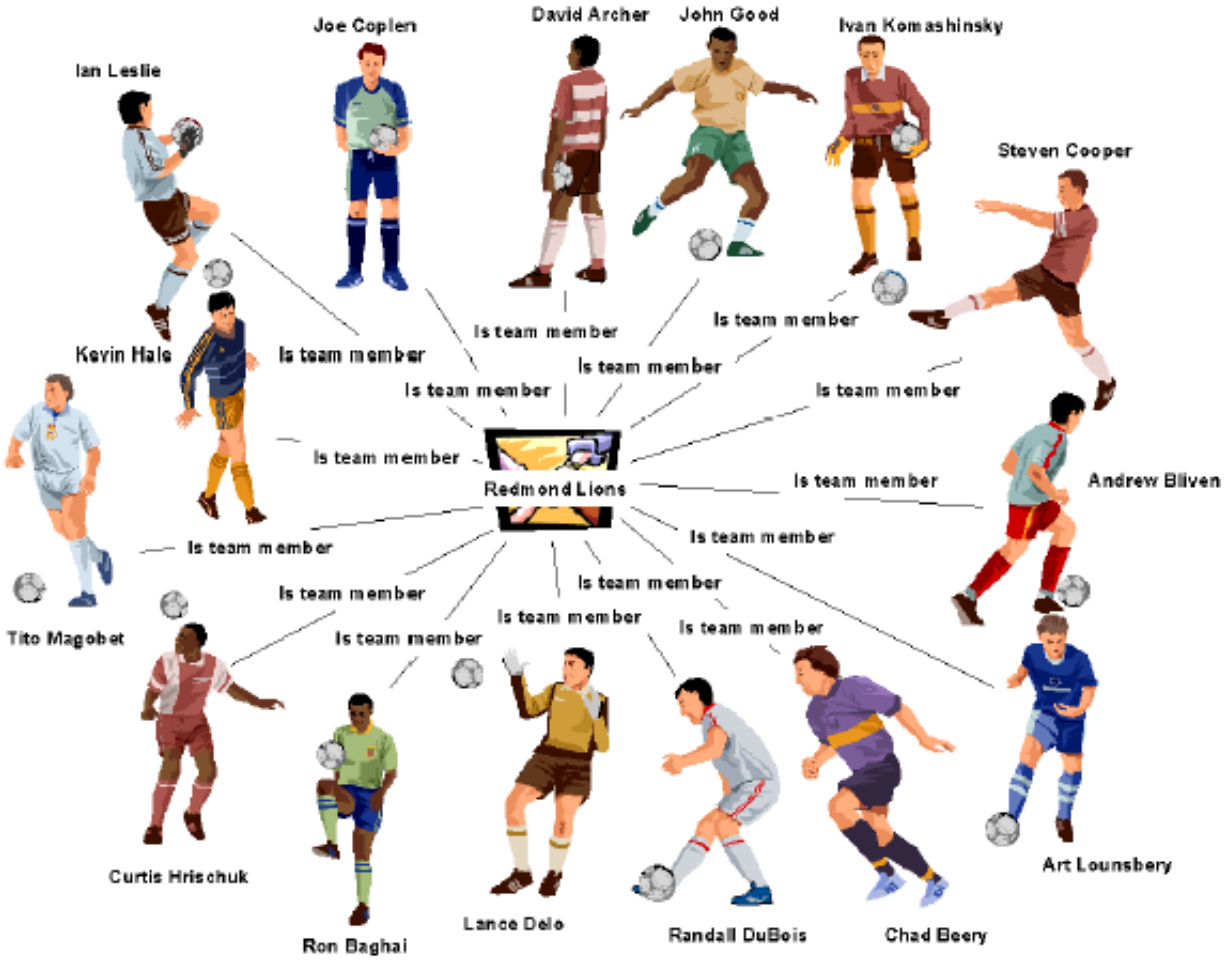
其他的基数还有 Not Specified 或 Specified By a Variable。Not Specified 基数没有限制。基数 Specified By a Variable （大部分为M 或N）同样没有基数限制。

当一个关系中的参与实体的上限和下限不同时，我们为上限和下限指定一对值，用括号括起来，之间用逗号分开，如（M，N）。根据上限的不同，可选的关系可以用（0，1）或（0，N）表示。

比如，足球队和球员的关系为（11，18）。实体 Redmond Lions 足球队与实体类型球员之间建立了一种关系，它由 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 和 Ian Leslie 组成。

图 5 足球队和球员之间的关系规定这种关系只有在球队球员在 11 到 18 之间时才有效

图 6 有效的关系是 Redmond Lions 足球俱乐部和 15 名球员之间的队员



足球比赛中经常有队员由于不当的行为或者犯规而领到黄牌或红牌。它们用实体类型 Cards 来表示。Cards 实体类型与基数为（0，N）的球员构建了一种 Received 关系。这意味着球员 David Archer 可能与三个 Card 实体有着 Received 关系，而 Lance Delo 却一个也没有。

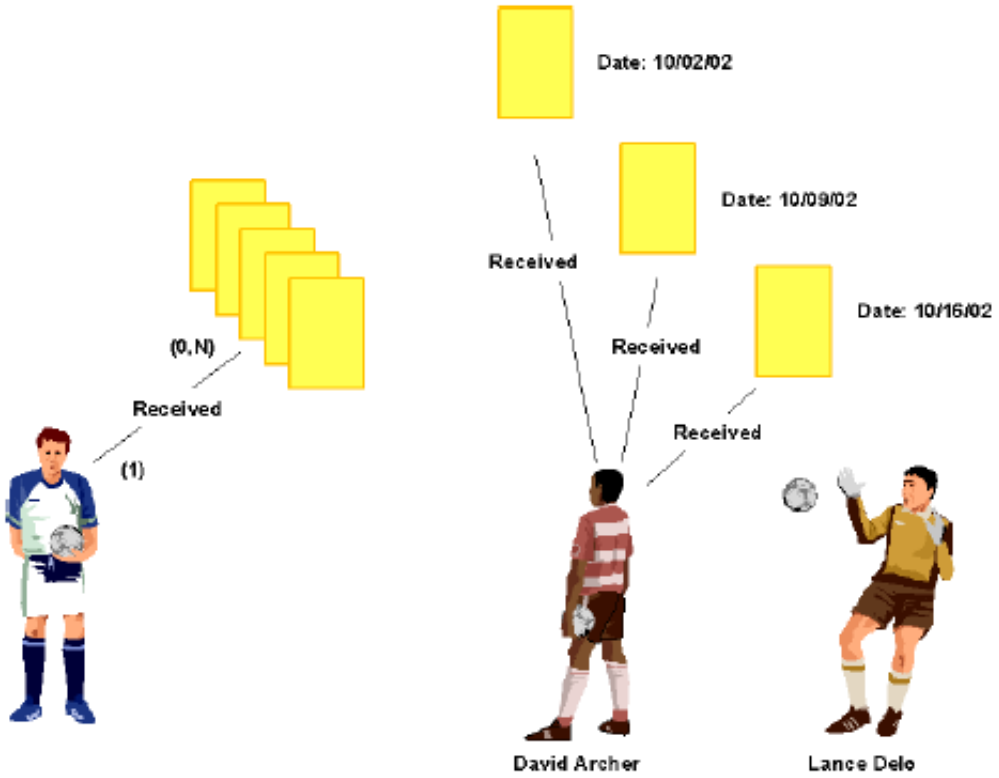


图 7 实体类型球员可能收到 0 个或者任意数目的 Card 实体类型：David Archer 收到 3 张牌，而 Lance Delo 一张牌也没有收到。

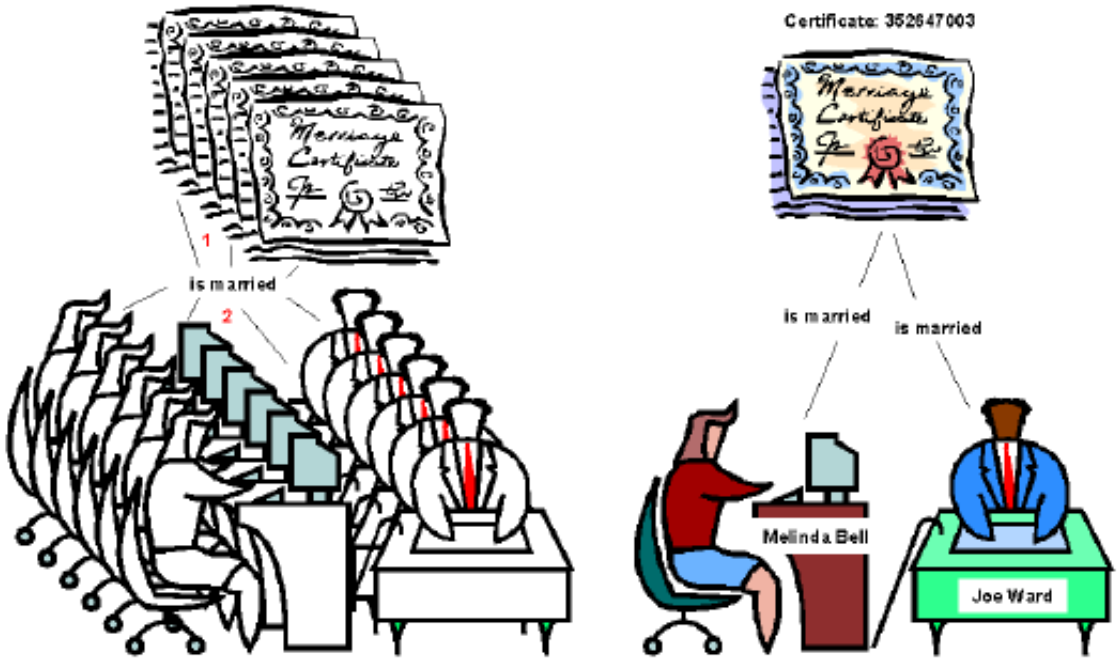
依赖关系

依赖关系指的如果在一个关系中指定的其中一个实体不存在，那么另一个实体也没有存在的意义。当依赖型实体（子类型）的各个实体依赖于超类型中对应父实体存在时，则一个实体类型依赖于另一个实体类型。

必须通过显式地定义一个下限不为 0 的基数，或者定义个一个不为 0 的固定基数，来指定一个强制的依赖关系。 实体类型结婚证就是两个实体间依赖关系的一个例子，其中结婚证依赖于实体类型人。关系是结婚，并且结婚证和两个人之间具有固定的依赖关系。

比如，实体结婚证 352647003 与实体 Joe Ward 和 Melinda Bell 具有固定的依赖关系。这意味着如果 Melinda Bell 或 Joe Ward 脱离该关系，那么至少从数据的角度来说，结婚证352647003就已经失效。

图 8 建立在实体类型人之上的依赖性实体类型结婚证，以及实体结婚证 352647003 与 Melinda Bell 和 Joe Ward 间的依赖关系



特化和泛化

核心 ER 模型只定义了实体类型间的基本关系。虽然利用基本的实体和关系就可以很容易地表示商业机构中的大多数简单数据结构，但是技术应用要求基于实体类型间的相似点和不同点的更复杂的结构。

特化和泛化
特化和泛化的目的在于重用与实体类型关联的属性和行为。

特化用于定义代表一个大型实体类型的一个特定部分的实体类型。特化后的实体类型从父实体类型继承了结构和行为，比如业务规则。然而，虽然特化后的实体类型扩展了父结构或类型，但是这决不是说它小于父类型。

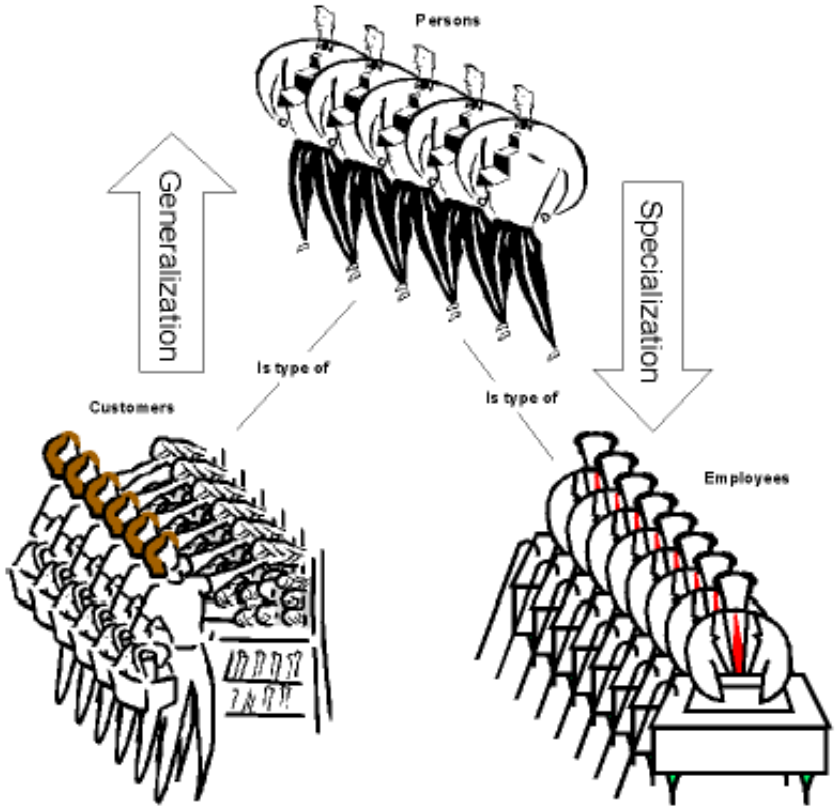
比如，Employee 是实体类型 Person 的一个特化，它需要实体类型 Person 的所有属性和关系。另外还有一种叫做 Customer 的实体类型，它也是实体类型 Person 的一个特化。这两种实体类型都具有 Person 的属性，它们被看作 Employee 或 Customer 的属性。因此在我们看 Customer 时，看到的是在实体类型 Person 和实体类型 Customer 中指定的所有属性。

泛化是正好相反的工作流。泛化实体类型（或者父类型）代表所有子类型的共同结构和行为，并且包含了来自子实体类型的所有共同属性。子实体类型具有父属性的所有内容，并且还拥有自己的属性。

泛化过程找出共同结构并在父实体类型中抽象出来。父实体类型通常在比较实体类型和简化模型时的重构阶段被找到。

虽然泛化仅利用面向对象或者对象关系数据库就可以直接实现，但是泛化也可以通过使用外键的任何关系数据库直接实现。

图 9 实体类型 Customer 和实体类型 Employee 被泛化为人实体类型；反向过程就是特化



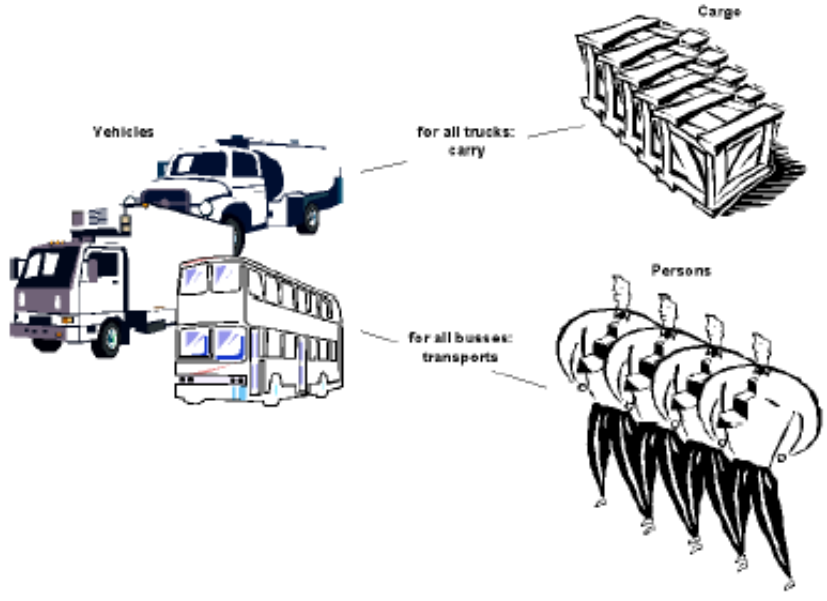
分类（Categorization）

特化是在实体上完成的，而分类则定义了关系类型上的约束条件。大多数情况下，分类是排他性的，这意味着根据实体状态的不同，一个实体要么参与关系 A 要么参与关系 B。该状态可能是一个属性值（另一种关系的存在），或是某些外部状态。

分类不改变实体的属性。它需要数据访问和操纵，来考虑分类中指定的约束条件。

交通工具就是分类的一个好例子。根据交通工具种类的不同，我们需要构建不同的关系。对于卡车，我们需要货物信息，而对于公共汽车，我们需要乘客姓名。这些信息将被用于不同的关系中，以便为这些关系提供有意义的上下文。

图 10 根据范畴的不同，卡车和公共汽车中的分类与实体类型 Cargo 和 Person发生关系



ER 方法学的表示法

目前，ER 建模中使用了数种表示法，包括陈氏 ER、Barker ER Information Engineering (IE)和 IDEF1X，其中大部分还包含关系表示法。

对象角色建模（ORM）表示法也是 ER 方法学的一员。它能表达非常精确和完整的业务规则，并且可以用图形阐述约束条件。不幸的是，这种详细程度需要具有丰富细节的大量图表。与 ORM 相关的问题是我们是否需要 ER 模型级的表示法精确度。ORM 表示法还很复杂，并且与其他表示法完全不同，因而那些没有使用 ORM 的项目组成员很难理解它。

UML 是一种表示法语言，最初用于软件设计，目前软件设计已经扩展到业务和数据库设计。UML 包括从分析到实施再到部署指定任何事项所必需的元素和图表。通过使用几种图表和数十种元素，UML 还能表达不同程度的系统抽象。这是一项非常独特的功能。但是，我们不需要知道 UML 的所有细节或者成功利用 UML 进行 ER 建模所需的所有视图和表示。

为了更好理解 UML 在 ER 方法学中扮演的角色，我们将描述 UML 如何处理前面已经提到过的 ER 建模的核心元素。

UML 中的实体类型

如前所述，实体类型标识了具有同样结构的一系列工件。实体类型是一幅蓝图，根据它能生成只能通过身份和状态互相区别的任意数目的工件。

UML 中的相应元素是类。根据定义，类能够隐藏内容，而实体具有可访问接口。这看上去互相矛盾，但是实际上并非如此。UML 允许类利用公共属性使结构公共化。

类一般用矩形表示，该矩形最多可分为三个部分：

第一部分包括类的原型和名称。原型指的是 UML 中为了强化共同特征而进一步进行的元素分类。比如，所有可能带有遗留原型的遗留类，可能将立即将遗留原型划分为不可修改的一类。虽然类本身是类型的一种表示，但是我们用原型<<实体>>来划分类型（<<…>>是用于指定原型的语法）。

第二部分包含具有类型和可见性的属性。它还可以包含属性的其他细节，比如初始值和原型。第二部分在缩略图中可以省略。

第三部分是为类的行为保留的。由于实体类型不需要行为，所以我们就略过该部分。

根据抽象级别的不同，类可以用一个、两个或三个部分显式。

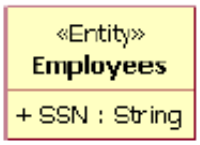
实体是实体类型的一个实例。在 UML 中，对象是类的实例。这意味着实体本身与对象相对应。

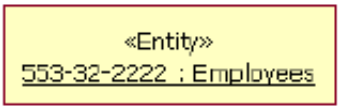
对象的表示来源于类的表示。最显著的区别在于对象的名称有下划线，并且只有一个或两个部分。

第一个部分包含以冒号隔开的可选的原型、对象名称，以及派生类的名称，之间用冒号隔开。至少必须指定其中一个名称。第二部分包含相关属性以及它们的值。

表示对象的最好的方法就是只使用一个部分，并将标识符 指定为对象的名称。

图 11 实体类型 Employees 和实体 553-32-2222 在 UML 中被显示为类和对象



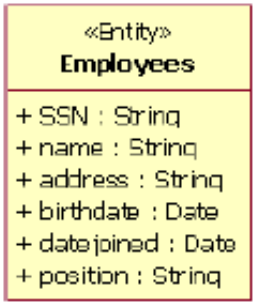


UML 中的属性

实体的相关状态和信息被作为对象的属性存放。实体类型的属性对于其他实体类型必须是可见的或者公开的。在类规格说明书中属性是按照可见性、名称和类型指定的。属性的类型为分析类型。它在设计阶段可能发生变更。

属性是在类的第二部分指定的。它们包含对于实体总是"+"（公开）的可见性规格说明书。名称与类型之间用冒号隔开。

图 12 具有公开属性（社会安全号码、姓名、地址、出生日期、加入日期和职位）的实体类型 Employees

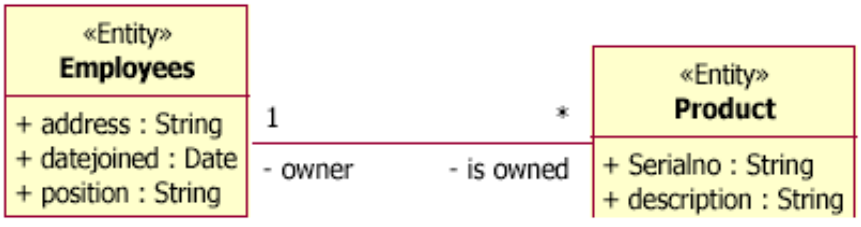


UML 中的关系类型

在 UML 中类之间的关系是为类型而非实例指定的。关系双方上的数字指定了基数：参与该关系的可能实例的数量。

关系的名称是直接在关系行中指定的，用于标识该关系。它有助于读者理解存在这种关系的原因。如果没有为一个关系指定名称，那么角色名将用于帮助读者理解该关系。下图可以这样理解：Products 被 Employees 拥有或者 Employees 是 Products 的拥有者。该关系描述中单词的单复数通过基数确定。

图 13 实体类型 Employee 和 Product 之间的关系



实体类型中用到的数据类型没有被标准化。用户可以使用所需要的任何数据类型。创建具有所有数据类型的术语表是一个不错的做法，因为它可以在公司内多个设计者和项目中实现标准化和可理解性。

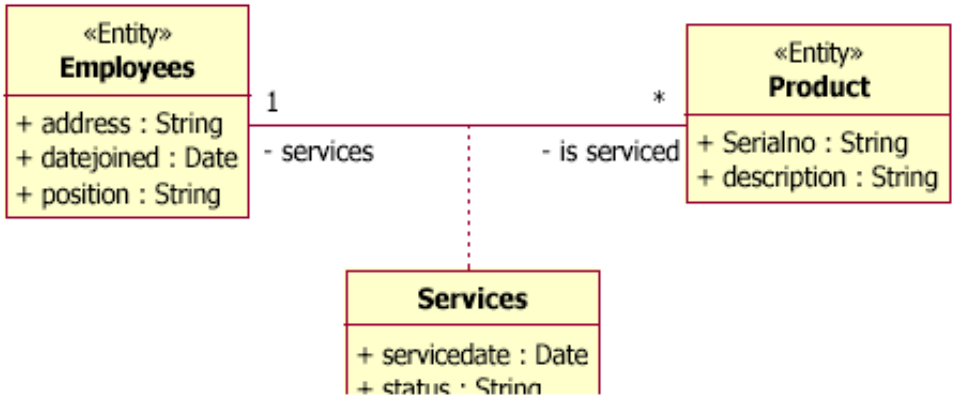
UML 中关系类型的属性

关系可以具有属性，这些属性显示在 UML 关联类中。关联类显示在一个矩形中，并且包含该关系的公共属性清单。关联类利用虚线与关系连接。不需要原型来解释类用法和为该类分类，因为附件已经对它做了定义。

关联类中的属性是用可见性、名称和类型指定的。

尽管看上去关联类、附件和关系是独立元素，但是它们实际上代表了同样的元素。关系和关联类的名称必须相关。

图 14 关联类中指定了关系类型 Services 的属性



UML 中的简单约束条件

基数

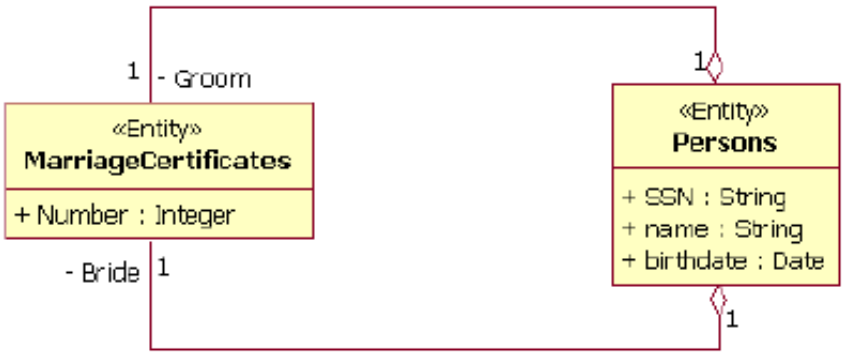
UML 定义了一种一致的方法来指定基数。它总是被关系双方上的数字指定。可能的定义包括用于一定数量（也可能是不限量的）实例的指定基数的单个数字，以及一对规定了基数的范围的以".."隔开的数字。用于无限基数的符号是"*"，它可以单独使用标识可选的无限关系，也可以与另一个很低的值结合使用，来指定强制关系（如"1..*"）。基数的下限和上限值可以是任意正数或者"*"，但是第一个数字必须小于或等于第二个数字。

图 15 实体类型 SoccerTeam 定义了与 11 到 18 个球员的实体类型 Player 的 plays 关系

依赖关系

UML 可以分辨两种形式的实体类型间的依赖关系。聚合是需要依赖性实体类型的两个实体类型之间的一种依赖关系。UML 中聚合的语法是在聚合方用空心菱形表示。同一方还有一个值为 1 的强制基数，它可以省略。

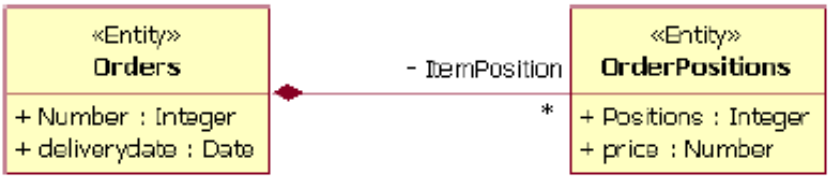
图 16 每个结婚证实体类型都依赖于扮演新郎和新娘角色的两个人



当聚合不是对于所有依赖关系都唯一，并且不是所有依赖性实例都必须与同一个实体相关时，就可以使

用聚合。当聚合是所有依赖关系的唯一实体时，UML 指定了一种叫做组合的强依赖关系。组合被表示为聚合方上的一个实心菱形。当聚合包含下级实体类型时会使用这种关系。

图 17 实体类型 OrderPositions 完全由组合指定的实体类型 Orders 定义



基数可以与聚合和组合一起使用，以便定义这些关系上的约束条件。

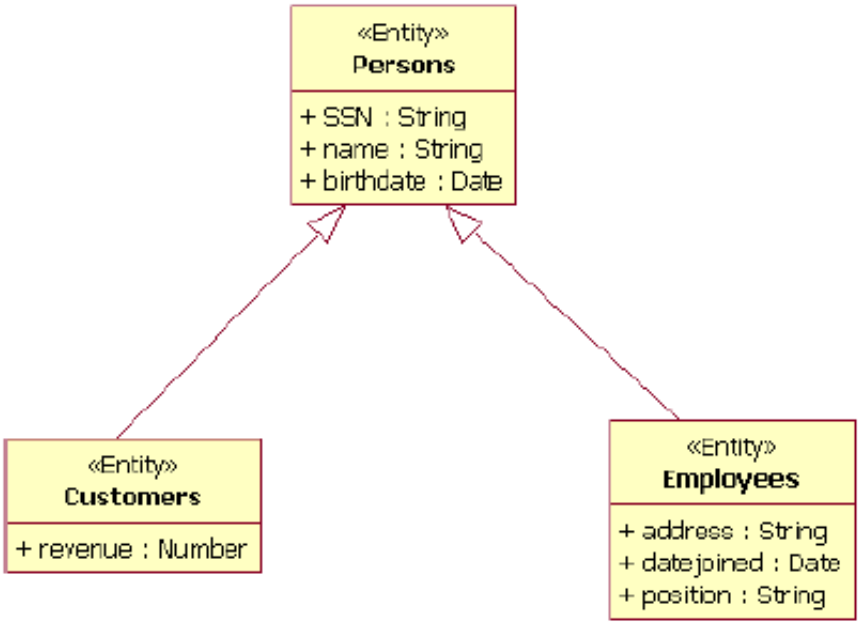
特化和泛化

实体类型的相同点和不同点分析是必不可少的一部分。特化降低了风险，并通过从父方那里继承需求从而减少了需求匮乏。泛化简化了系统中实体类型的模型和实现过程。

在 UML 中泛化是在关系的父方用空箭头表示的。泛化不是两个实体类型间的一种关系。它是从特定实体类型到一般实体类型的派生。在泛化关系上不允许有基数。

父实体类型的所有属性和关系被特化后的实体类型继承。到其他实体类型的属性和关系不能从特化实体类型中删除。

图 18 泛化将实体类型 Employees 和 Customers 定义为继承了 Persons 属性的实体类型 Persons



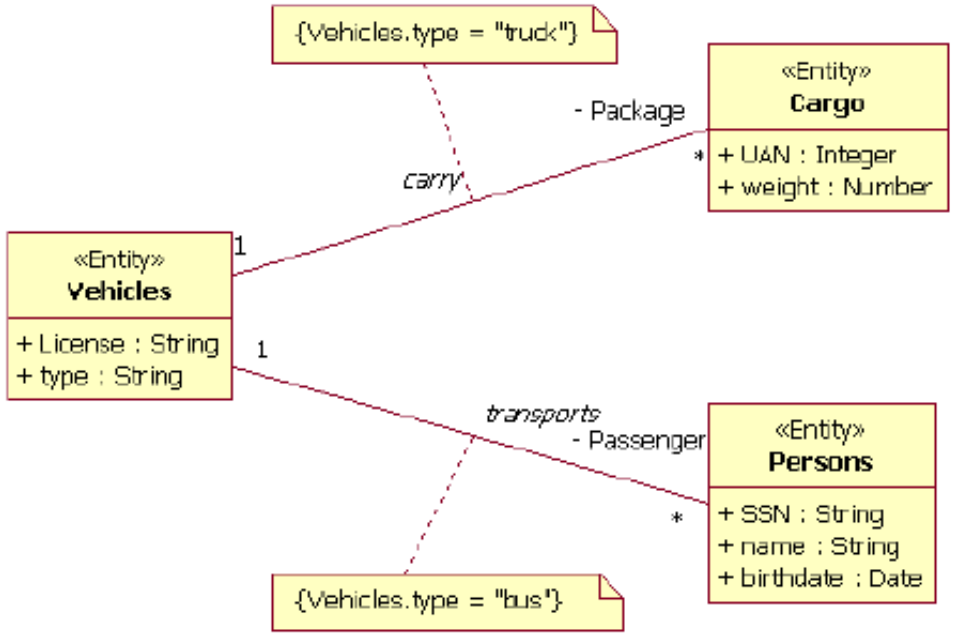
分类

同一实体类型的范畴规定了实体如何根据它们的特征（比如所有员工都缺勤）互相相关。分类的语法使用了在 OCL（对象约束语言）中指定的与该关系连接的约束条件。

约束条件旨在指定动态行为。当约束条件被评估为有效时，这些关系也是有效的。

通常约束条件是互相排斥的，可以利用互斥关系之间的约束条件{xor}为约束条件建模。

图 19 实体类型的分类定义了关系类型的标准--Cargo 对于卡车类型的 Vehicle 很重要；Person 可以利用"公共汽车"类型的 Vehicle 被运输



结束语

与一般的观点不同，ER 方法学并不仅限于关系数据库的开发。笔者不得不同意在大多数情况下，设计流程的输出会在关系数据库中实现；然而，这并不是它的一个条件。ER 方法学的焦点在于基于工件的设计。它输出的是互相有关系的工件（实体类型），以及澄清了实体类型和关系的约束条件。该输出用于创建具有额外技术相关约束条件的关系模型。

ER 方法学用于工件驱动系统的分析和设计。它可用于概念和逻辑上的建模，但本意并非是为了物理设计。ER 方法学只描述了工件的静态视图，而没有描述系统的动态。

为了创建一个平滑的开发流程，开发团队的所有成员需要"操同一种语言"，这一点很重要。这是因为对信息的曲解可能导致延误，造成意想不到的错误，并降低团队成员的总体效率。UML 通过提供一种很容易被系统开发团队所有成员理解的标准化语言消除了这些担忧。

IBM软件集成方案

IBM Rational 支持 IBM 软件公司的其他大量产品。IBM 软件解决方案给予实现优先级业务和 IT 目标的强大动力。

- DB2® 软件利用数据启用（data enablement）、数据管理和数据分布解决方案帮助您充分利用信息。
- Lotus® 软件利用授权、管理、交流和共享知识方面的解决方案帮助您的职员提高生产力。
- Tivoli® 软件帮助您管理运行电子商务基础构架的技术。
- WebSphere® 软件帮助您将现有的业务关键流程扩展到 Web 上。
- Rational® 软件利用工具、服务和最佳实践帮助您提高了软件开发能力。
- 请参加 Rational Club关于本文的讨论。

来自 IBM 的 Rational 软件

来自 IBM 的 Rational 软件通过提高各个公司的软件开发能力帮助他们创造业务价值。Rational 软件开发平台集成了软件工程中的最佳实践、工具和服务。借助于该平台，公司可以响应更快、更具弹性和更专注，从而在当前随需应变的世界中脱颖而出。Rational 基于标准的跨平台解决方案帮助软件开发团队创建和扩展业务应用程序、嵌入式系统和软件产品。财富 100 强中的 98 家都使用 Rational 工具来提高软件开发的速度和质量。如需查阅更多信息，请访问 www.rational.com 和 Rational 的电子月刊 www.therationaledge.com。

条评论

请 登录 或 注册 后发表评论。

添加评论:
注意：评论中不支持 HTML 语法

有新评论时提醒我

剩余 1000 字符