一种生成动态BOM的优化方法的研究

专　　业：计算机软件与理论
硕 士 生：侯恩超
指导老师：李磊 教授

摘要

在 ERP(Enterprise Resource Planning)实现中，通过物料清单(BOM)配置产品结构是非常重要的一个方面。一个 BOM 描述了产品的组成结构，它通常以层次化结构在关系数据库中实现。这些描述包含了终端产品、装配件和物料之间的关系。在 ERP 或者产品数据管理(PDM)系统中实现这种结构的传统方法是为每一个产品模型设计一个 BOM。这在"强调以客户为中心，提供个性化的服务"的市场环境下，难以实现满足客户个性化条件下的产品规模生产。

本文综合考虑产品 BOM 的设计和客户定制，结合 BOM 模型研究中的现有成果，提出了一种基于对象模型和关系实现的动态 BOM 的生成方法。为 BOM 建立一个面向对象的模型，采用特定的对象关系映射策略，将对象的物料清单映射到关系数据库中。这样既可以充分利用面向对象的方法来描述和建立物料清单的目的，又可以采用成熟的关系数据库方法来存储和管理 BOM 数据。本文提出了对 BOM 进行动态建模的思想，通过元类对 BOM 的结构进行管理，使 BOM 的结构和层次间关系相分离，进一步降低了 BOM 的耦合度，很好的满足了企业对 BOM 定义的个性化定制和动态更改的需求。

关键词：物料清单，动态建模，元模型，对象关系映射，持久化

The Study of an Optimum Method of Generating Dynamic BOM

Major: Computer Software & Theory

Name: Hou En chao

Supervisor: Li Lei Prof

Abstract

In the realization of ERP, it is an important part to configure product structure through BOM. A BOM describes the structure of product, we usually realize it with hierarchal structure in the relational database. These descriptions include the relation of end products, assembled parts and material. In ERP or PDM system, the traditional method to realize this kind of structure is to design specified BOM for each product model. It is hard to meet customers's personalized needs, at the same time, start large scale production in the market circumstance, which customers are the focus and personalized service is needed.

In this paper, we consider both design of products and customers customization, combine with the exsisting production of BOM models. Then, we put forward an optimum method of generating dynamic BOM which mapped the BOM of Objects to relational database by special mapping strategies. It can not only describe and construct BOM by object-oriented method, but also store and manage data of BOM by mature methods of relational database. The paper also put forwards the thought of dynamic modeling for BOM. We can manage the structure of BOM with a meta-class. It separates hierarchy relation from structure of BOM. As a result, the degree of coupling is lowered. It satisfies enterprises with personal customization of BOM definition and demands for dynamic modification.

Key words: bill of material, dynamic modeling, meta model, object/relation mapping, persistent

原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：侯恩超

日期：2008 年 5 月 24 日

使用授权声明

本人完全了解中山大学有关保留、使用学位论文的规定，即：学校有权保留学位论文并向国家主管部门或其指定机构送交论文的电子版和纸质版，有权将学位论文用于非赢利目的的少量复制并允许论文进入学校图书馆、院系资料室被查阅，有权将学位论文的内容编入有关数据库进行检索，可以采用复印、缩印或其他方法保存学位论文。

学位论文作者签名：侯恩超　　导师签名：

日期：2008年5月24日　　日期：2008年5月24日

第1章 引言

1.1 研究背景和意义

BOM，即物料清单，也称为产品结构，产生于产品的设计阶段，主要表达了企业产品的结构和产品中零部件的构成关系，是一个典型的树状结构[1]。BOM 是广泛使用在制造系统中的一种用于表达产品信息的方式。它的产生是通过对产品结构相关对象的属性提取后得到的，因此产品结构是 BOM 数据的主要来源。BOM 信息是销售、计划、设计、生产、供应、物料、工艺等部门都需要参考的重要文件，是实现数据共享和信息集成的关键数据。

现有的 PDM(Product Data Management)通常是基于单个产品的 BOM 进行管理。在大规模定制环境下，为满足客户的多样性需求，需要不断增加系列产品的品种。在这种系列产品中，不同产品的 BOM 结构存在相似性，且包含大量的相同零件。随着系列产品品种的增加，导致 BOM 结构的冗余。在面向客户的大规模定制的生产模式下，如果采用基于传统 BOM 的产品数据管理方法来管理品种繁多的产品数据，将导致冗余加剧和管理低效率。

客户个性化的大规模定制已经成为企业面临的一项重要课题，也是产品提供方与消费者之间正在建立的一种新型关系[2]。因此企业产品设计和生产组织都必须适应这种变化，这就对 BOM 的构造提出了更高的要求，主要体现在如下几个方面：

- 客户定制的快速响应
- 网上定制的可支持性
- 较高的运作速度
- 产品结构的直观、形象化
- 产品设计的客户需求化

由于 BOM 在企业产品中占有非常重要的地位，一直以来都是研究的重点，也产生了许多良好的 BOM 构造方法。本文在对这些方法进行分析以后，借鉴其中的一些思想，提出了基于面向对象技术的动态 BOM 模型，并对其性能和可行

性进行了分析。

1.2 BOM 模型的研究现状

多年来各国研究人员对 BOM 模型进行了深入而富有成效的研究和开发工作。主要包括以下几个方面：

1.2.1 BOM 的构造模型

1.2.1.1 传统 BOM 模型

传统的 BOM 是在关联数据模型的支持下实现的，数据以表的形式存放，表里的数据类型是数据的一种简单逻辑模型，简洁和数据独立性是其主要特征，但由于它在某些应用中的缺陷，在表达复杂对象的语义时会比较困难。目前，绝大部分的研究和开发都是针对传统 BOM 展开的。

1.2.1.2 面向对象数据模型

面向对象数据模型继承了关联数据模型和语义数据模型的优点，并用面向对象语义来体现和实现，它支持数据与操作的封装、抽象与具体的关联、整体与部分的关联等概念机制，具有丰富的语义表达能力，已达到对问题域同样高层次的自然理解[3]。面向对象数据所具有的丰富的语义和抽象机制为 BOM 的建模与设计提供了新的手段。从面向对象看，每一种产品都能以类的形式来抽象，每一个零件也能以类的对象形式来表达，对象之间通过 Is-Part-Of 关联成复杂对象，通过 Is-A 关联抽象为类。

J. Rumbaugh，M. Blaha，W. Premerlani，F. Eddy 和 W. Lorensen 提出了面向对象建模和设计的思想，这是面向对象技术初次使用在关系数据库中。曹礼廉和李芳芸提出了 BOM 的面向对象的建模与设计方法[4]，并在此基础上探讨了面向对象 BOM 在 CIM 中的应用；王立松等针对 BOM 的特点，为 BOM 建立了一个面向对象的模型，并采用了特定的对象关系映射策略将该面向对象的 BOM 模型映射到关系数据库中；李向东和范玉清针对产品结构与配置管理问题，利用面向对象技术，以单一产品数据源作为底层数据支持，给出了 BOM 的对象模型[5]，

并将其应用于系列化包装机的动态产品结构与设计流程的有序管理中；此外，董桂岩，郭旭红等也对 BOM 的面向对象模型进行了比较深入的研究，并提出了许多实用的策略。

1.2.1.3 通用 BOM 模型

通用 BOM(GBOM,Generic Bill of Material)，描述了所有部件的合法变量。对某个部件的描述是独立于特定部件组成的产品的，这种抽象使得部件可以在多个产品中复用。这种通用的结构可以为一个系列的产品提供一个 BOM，部件之间的差别通过部件所含有的变量来体现。当用户选择一个指定的变量，系统会遍历通用 BOM 结构，给出必要的详细说明，就可以得到一个指定的 BOM。这样，只要添加一些限制，就可以简化部件的变量。所有产品通用一个结构，大大简化了对 BOM 模型的维护。

KAI A. OLSEN，PER SAETRE 和 ANDERS THORSTENSON 提出了一种面向程序的 GBOM[6]，它将 GBOM 以程序的形式进行存取，加快了重构 BOM 的速度，提高了现有 BOM 的管理效率；这种图便于呈现产品的合法变量，因此支持变量描述处理。VanVeen 提出了一种 GBOM 的结构[7]，这个结构允许父亲对儿子的变化形式加限制；同时，它又考虑到部件的描述独立于部件在高层次产品中的使用方式。

1.2.1.4 BOM 赋权有向图模型

还有一种较有影响的模型是基于 BOM 的赋权有向图模型。传统的 BOM 都是由树形结构来表示的，全部产品和零部件都以父件或子件的形式连同他们之间的所属关系一起表示成产品结构树。然而，树形结构只是产品 BOM 结构的上层表现。由于节点在树形结构中可能在相同或不同层次重复出现，树形结构不便于表达这些节点之间的关系。产品 BOM 赋权有向图模型是一种通用的 BOM 模型，以 BOM 最本质的属性建立模型，方便的实现了关系数据库的规范化操作，克服了传统产品 BOM 中“一子多父”情况下的困境，可以实现 PDM/CAD/CAPP 与 ERP/MRPII 系统的集成，使不同部门的或系统的信息和资源得到了最大程度的共享，使不同版本的 BOM 的准确性、完整性、一致性和通用性均得到很大提高[5]。

在这方面的研究中张国军，邵新宇提出了一种 PDM/CAPP/MRPII 的集成方案。石为人提出了一种产品 BOM 赋权有向图通用模型[8]，给出了其相应的关系数据库实现，研究了产品 BOM 赋权有向图模型的维护方法，讨论了基于企业实际的模型树形扩展，并给出遍历和搜索的思想、相关算法以及编程语言实现方法。

1.2.2 BOM 的动态配置

与传统 BOM 相比，可配置 BOM 模型是一种动态的产品构造维护和设计模型，其主要特点是产品部件的动态不依赖性及对客户定制反应的快速动态性。目前，主要采用模块化设计和参数化配置来实现 BOM 的动态配置[12]。

模块化设计一般用于由许多通用件制成的并有许多种组合的复杂产品。在产品 BOM 中，对于抽象出来的模块节点，只要保存其模块代码，在 BOM 实例展开时再调入该模块。这种设计方法既为客户提供了较广的选择范围，又大大的降低了记录的存储和维护费用。

参数化配置[10,12]是指对于同种型号的产品系列，首先构造出不带参数的产品族，然后再对该产品族进行参数化配置。通过使用参数化，使得产品的系列化并不会带来众多庞大而又类似的 BOM，同时，也能够规范产品参数数据，对企业的管理具有重要的意义。

WolfraMWoo 提出了一种规则驱动的变量零件和变量 BOM 的生成机制[11]。它的思想是存储某产品族和相应的规则，当客户提供需求说明时，利用已有产品族结构，根据需求查找存在的规则来实时的生成满足客户需求的产品，这样就不用再去管理大量的数据，节省了大量的存储空间。J.C. Hern´andez Mat´ıas 等人提出了一种基于变量说明属性模式的 BOM 生成法方法[12]。其思想是通过提取客户的定制信息和对定制属性的划分实现了 BOM 的动态生成，使客户进一步的从录入参数的复杂劳动中解脱出来。钱炜等结合浙江某企业的 PDM 项目，对如何从产品族生成物料清单的产品配置方法进行了研究，通过建立产品编码、产品族结构、物料属性之间的多重关联，实现了根据用户需求形成产品特征，再根据产品编码自动从产品族中生成特定物料清单的设计方法[13]；王庆国等针对客户定制这一课题，提出了动态 BOM 的设计模型及基于数据窗口与 SQL 的算法[14]，并

以某公司单相电度表为例，说明了其使用情况；此外，陈兵等针对电子行业 BOM 的复杂性，也探讨了 BOM 在该行业的构造和管理方法，为 BOM 的动态配置提供了实用的设计方法。

1.3 本文的研究内容

本文的研究内容主要是：

1、对现有的 BOM 模型进行分析

通过对现有的 BOM 模型进行分析、比较，把每种模型的优缺点清晰的展开，抽取出其中一些优秀的思想，针对存在的问题，寻找切入点，希望能将两种或以上的模型有效地融合，实现优势互补。对现有模型的理解的深度，直接影响到改进思想的先进度，也是激发出一种较好模式的动力源泉。

2、提出一种面向对象的动态 BOM 模型

该部分介绍了面向对象的它动态 BOM 模型的构建思想，这其中有改进、有创新。首先，把动态建模思想引入到面向对象模型的构建之中，分析了模型的数据组成；其次，给出了一个 BOM 的形式化定义，更加严谨的剖析了 BOM 的组成要素；最后，给出了一个 BOM 的元模型，将模型性的结构以面向对象的形式展现出来，这是本文的重点。

3、对模型的实际应用效果进行分析

该部分基于动态 BOM 模型构造了一个小型的系统，着重介绍了对象关系映射策略，证实了新模型应用的可行性，并把该模型与现有模型进行了比较以说明其优缺点。

第2章 BOM 构造方法的相关研究

2.1 BOM 的研究

2.1.1 BOM 的定义

物料清单(BOM, Bill of Material)，是非常关键的基础数据，它用来描述产品组成结构，即描述了制造产品所需要的原材料与零件、部件、组装件之间的从属关系[10]。它是物料需求计划(MRP, Material Requirements Planning)的重要组成部分。在运行 MRPII 时，首先需要有一个能正确、完整地阐明产品结构的 BOM，它是物料需求系统的主要输入之一。

2.1.2 BOM 的分类

BOM 可以分为两种类型：静态(又称固定或者传统)BOM 和动态(参数化)BOM[15,23,25]。下面具体介绍这两种 BOM 的特点。

2.1.2.1 静态 BOM

静态 BOM 是这样一种零件的 BOM，这种零件是按照常规的方法，并由相同的组件、工序和原材料制成。典型的，我们会为标准组件和标准部件创建静态 BOM。

静态 BOM 比较易于构造，新手比较容易上手，创建者也不需要对潜在的产品族做全面的分析。因此，虽然静态 BOM 的处理方法比较原始，但是仍然现在的企业中被广泛的使用。

静态 BOM 在制造型企业中非常常见，如为一把标准的椅子创建一个 BOM。这个 BOM 专属于这一种椅子，只要它有一个属性发生改变(比如椅子的颜色)，就需要为这种新的产品重新构造 BOM。因此，我们就可以想象，如果某种商品的种类比较繁多时，就会造成大量的数据冗余，BOM 的维护也比较困难。

2.1.2.2 动态 BOM

动态 BOM 是这样一种产品或零件的 BOM，它们的尺寸、颜色和其他的一些可选件都是可以选择的。你对尺寸或者选项的选择会影响低层组件和工序，你还可以控制除操作和组件以外的选择。

动态 BOM 实际上是一种综合性产品结构维护和设计模型，之所以称之为动态在于其子结构部件的动态性划分、产品部件的动态不依赖性以及产品层次结构直观清晰基础上的维护动态性、客户定制反应的快速动态性[2]。它是为了解决传统 BOM 的数据冗余量大、定义困难和可维护性差的问题而提出的。

动态逻辑(参数化加上"可选件和特征件")一个通用 BOM 用于一整个的产品族而不用去考虑它的样式、颜色和尺寸等等。动态 BOM 主要由以下几种优点：

- 所有相似的产品你可以使用相同的动态(参数化)BOM。如果你要制造橱柜，一个 BOM 就可以满足所有样式的橱柜而不用去关心它的尺寸，门的数量或者样式。
- 如果你要改变一个构造细节(机器类型、加工速度等等)，你只需要处理一个 BOM，而不需要面对成千上万的 BOM。
- 尺寸要素可以用来指定改变工艺路线(机器的尺寸限制)，选择物料(根据不同坚韧度要求而改变厚度)，以及确定处理时间。
- 把变量指令传递到较低的层次。产品 BOM 的尺寸和其他的可选要素可以指导低层组件或者物料清单选择特征件或者加工路线。

由于 BOM 在 MRPII 中起到极其关键的作用，多年来一直是该领域研究的重点，其中大部分是关于静态 BOM 的。近年来，随着市场需求的不断变化，已经有越来越多的人致力于动态 BOM 的研究。两方面的研究都取得了丰硕的成果，分析它们的优缺点是我们研究的出发点，而其中的一些优秀思想又为我们的改进工作提供了参考。接下来就对 BOM 的相关方面的研究做一下介绍。

2.2 BOM 模型

BOM 用于将产品的结构信息存储于计算机中，而产品结构树用于图形化一个产品的组成结构。产品结构树形如一颗倒长的树，树根在上，树枝在下，它能

够非常直观的反映产品的结构[19]。其简单的表现形式如图 2-1 所示。其中，眼镜由 1 个镜框、2 个镜片和 2 个螺钉组成；镜框又是由 1 个镜架、2 个镜腿、2 个鼻托和 4 个螺钉组成；图的右边显示了各部件所在的层次，0 层为最高层，1 层其次，其他依次类推，数越大层次越低。

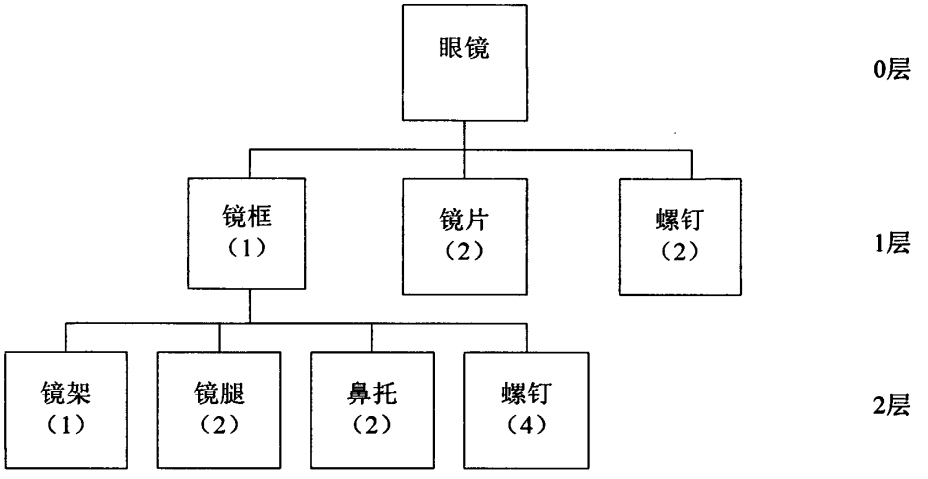

图 2-1 眼镜的产品结构树

产品结构树是产品的外在表现形式，而物料清单 BOM 则是产品的内在表示形式。在 BOM 中，它包含了一个产品在生产或装配时所需的各种组件、零件及原材料的清单，不仅反映了一个产品的物料构成项目，同时还指出了这些项目之间的实际结构关系，即从原材料、零件、组件、部件直到最终产品，每一层次间的隶属关系。例如，图 2-1 的产品结构树用 BOM 来表示，其表示形式如下表：

表 2-1 一副眼镜的 BOM

产品零件号：#2000-眼镜				
零件号	描述	装配数量	单位	层次
20100	镜框	1	个	1
. 20110	镜架	1	个	2
. 20120	镜腿	2	个	2
. 20130	鼻托	2	个	2
. 20099	螺钉	4	个	2
20300	镜片	2	个	1
20099	螺钉	2	个	1

可以发现，在 BOM 中，每一个关系都定义成“父项/从属子项”的形式，并给出了从属子项的数量。同时，一个关系中的某个从属子项也可以在其他关系中充当“父项”，从而形成了项目之间的层次从属关系。BOM 就其实质而言是一份反映产品结构的技术文件。在 BOM 中列出了构成每个上属父项的零部件和材料，以及它们之间的数量关系。显然，BOM 给出了两个最基本的重要信息：

1) 一个父项(产品、部件、组件等)是由哪些子项(零件、原材料等)所组成的。同理，也可以说明一个子项应属于哪些父项。

2) 一个父项对构成它的子项的数量要求。BOM 给出了每个子项在其父项中的需要量。

由上可知，一个 BOM 文件应该至少包含 3 个数据项：标识代码、需求量(单位父项所需该子项的数量)、层次码(该项目在结构表中相对于最终项目的位置)[30]。除此之外，BOM 中还可以包括单位、说明等。

通常，外在表现形式的产品结构树是不会改变的，但其内在表示形式(即物料清单 BOM)会根据用户的需求而有所改变。由于内在表示形式的合理与否直接影响到系统的处理性能。因此，根据实际的使用环境，灵活地设计合理且有效的 BOM 是十分重要的。物料清单 BOM 一般有如下常用的格式[10]。

2.2.1 传统的 BOM

1) 单层展开。也叫单层 BOM，单层分解表。

单层展开格式显示某一装配件所使用的下层零部件。它只表明直接用于父项中的那些组件以及为父项所要求的数量。上面提到的眼镜的单层 BOM 结构如表 2—2 所示。

表 2-2 眼镜的单级展开 BOM

父项	子项	装配所需数量
眼镜	镜框	1
眼镜	镜片	2
眼镜	螺钉	2

镜框	镜架	1
镜框	镜腿	2
镜框	鼻托	2
镜框	螺钉	4

通常，单层 BOM 可给出组件的描述，有时也给出一些附加信息，例如，单级 BOM 可以提供组件在工艺线路中的位置等一些参考标志。采用多个单层展开就能完整地表示产品的多层结构。大多数的 BOM 都是采用这种结构来构造 BOM。

2) 缩行展开。也叫多层 BOM，完全分解表，内缩式 BOM。

缩行展开格式是在每一上层物料下以缩行的形式列出它们的下属物料。同一层次的所有零件号都显示在同一列上，一般是缩排式的。如表 2-4 所示就是眼镜的多层 BOM 结构。

表 2-3 眼镜的多层展开 BOM

序号	父项	子项	装配数量	层次号
1	眼镜	镜框	1	1
2	眼镜	镜片	2	1
3	眼镜	螺钉	2	1
4	眼镜	镜架	1	2
5	眼镜	镜腿	2	2
6	眼镜	鼻托	2	2
7	眼镜	螺钉	4	2

缩行展开的格式是以产品制造的方式来表示产品的。多层 BOM 要比单层的复杂。它把单层 BOM 连接在一起，来表明直接或间接用于制造各级父项所有的自制件或外购件。多层 BOM 还列出了父项所需要的每个零件的数量。

3) 汇总展开。也叫综合 BOM，结构分解一览表。

汇总展开的格式列出了组成最终产品的所有物料的总数量。它反映的是一个最终产品所需的各种零件的总数，而不是每个上层物料所需的零件数。它可用于

快速估计完成一定数量装配的总需求，或用来估计一个组件中的子装配件变化对成本的影响。这种格式并不表示产品生产的方式，但却有利于产品成本核算、采购和其他有关的活动。以眼镜为例，其 BOM 如表 2-4 所示。

表 2-4 眼镜的汇总展开 BOM

产品	零件	需要数量
眼镜	镜片	2
眼镜	螺钉	6
眼镜	镜架	1
眼镜	镜腿	2
眼镜	鼻托	2

4) 单层跟踪。也叫单级反查表，单级回归表。

单层跟踪格式显示直接使用某物料的上层物料。这是一种物料被用在哪里的清单，它指出的是直接使用某物料的各上层物料。以眼镜为例，其 BOM 如表 2-5 所示。

表 2-5 眼镜的单层跟踪 BOM

零件	父项	需要数量
镜片	眼镜	2
螺钉	眼镜	2
螺钉	镜框	4
镜架	镜框	1
镜腿	镜框	2
鼻托	镜框	2

5) 缩行跟踪。也较多级反查表，完全回归表。

缩行跟踪的格式指出了某零件在所有高层物料的使用情况。它可查找直接或间接使用该物料的所有高层物料，在进行物料反查时这种格式很有价值。

6) 汇总跟踪。也叫汇总反查表，回归一览表。

它指出了某物料在所有高层物料中被使用的情况，可用于查找直接或间接使用该物料的所有高层物料直至产品。“所需数量”表示装配成该层次的物料所需的

零件总数。在决定生成某物料需求的上述物料以及评价工程设计变化的效果时，汇总追踪格式很有价值。

7) 末项追踪格式。

末项追踪格式又称末项反查表，它仅仅列出使用某个零件的那些末项。

2.2.2 矩阵式的 BOM

矩阵式的BOM是对具有大量通用零件的产品系列进行数据合并后的得到的一种 BOM。这种形式的 BOM 可用来识别和组合一个产品系列中的通用零件。其结构如表 2-6 所示。

表 2-6 矩阵式 BOM

零件号	产品型号		
	A1	A2	A3
x	5	5	3
y	1	2	3
z	3	4	#
w	2	#	4

在该表中，左面列出的是各种通用零件，右上部列出了各个最终产品，下面的数字表示装配一个最终产品所需该零件的数量。“#”表示该产品不用此零件。对于有许多通用零件的产品，这种形式的 BOM 很有用处。但矩阵式 BOM 没有规定产品制造的方式，它没有指出零件之间的装配层次，不能用于指导多层结构产品的制造过程。

2.2.3 加减 BOM

这种 BOM 有时又称为“比较式”，或“异同式”BOM。它以标准产品为基准，并规定还可以增加哪些零件或去掉哪些零件。假设以眼镜为标准产品，通过加减 BOM 来产生其变型产品眼镜/1，其结构如表 2-7 所示。

表 2-7 眼镜/1 的加减 BOM

序号	标准眼镜	需要部件数量	眼镜/1	需要部件数量
0	眼镜		眼镜/1	
1	镜框	1		1
1-1	镜架	1		1
1-2	镜腿	2	+1	3
1-3	鼻托	2		2
1-4	螺钉	4	-1	3
2	镜片	2		2
3	螺钉	2		2

一个特定的产品就被描述为标准产品加上或者减去某些零件。加减 BOM 能有效地描述不同产品之间的差异，但不能用于市场预测。

2.2.4 模块化 BOM

模块化 BOM(Modular BOM)是对通用型的产品族间进行模块化处理而简化形成的物料清单[10]。对于一些有类似结构特征的产品，如果都按照普通的物料清单来核算管理，则数据重复量很多，系统的效率降低，管理也不方便。而利用模块化清单，凡是用到该通用模块的均无须重新输入数据，只需引用该模块组件，大大提高管理效率。

模块化 BOM 通常用于系列产品的情况下。系列产品通常由 3 种类型的物料组成，下面还是以眼镜为例来说明。

1）通用件(General)

所有产品都必须用到相同的物料。如螺钉，是每副眼镜都不可缺少的部件。

2）特征件(Feature)

也即一组可选的基本组件，它是所有产品都不可少的组件，但是组件中有多种选择件，必须任选其一。如眼镜的镜架，每副眼镜都需要，材质有金属的，也有塑料的；颜色有银白、金黄和灰灰黑等，可供客户选择，但是必须选其中之一，不能不选。

3）可选件(Option)

可以包括在成套产品中，也可以不包括，即可选可不选。如用于稳定眼镜的镜把。

对具有这种特点的产品，应建立模块化产品结构。图2-2是眼镜的模块化产品结构的示意图。

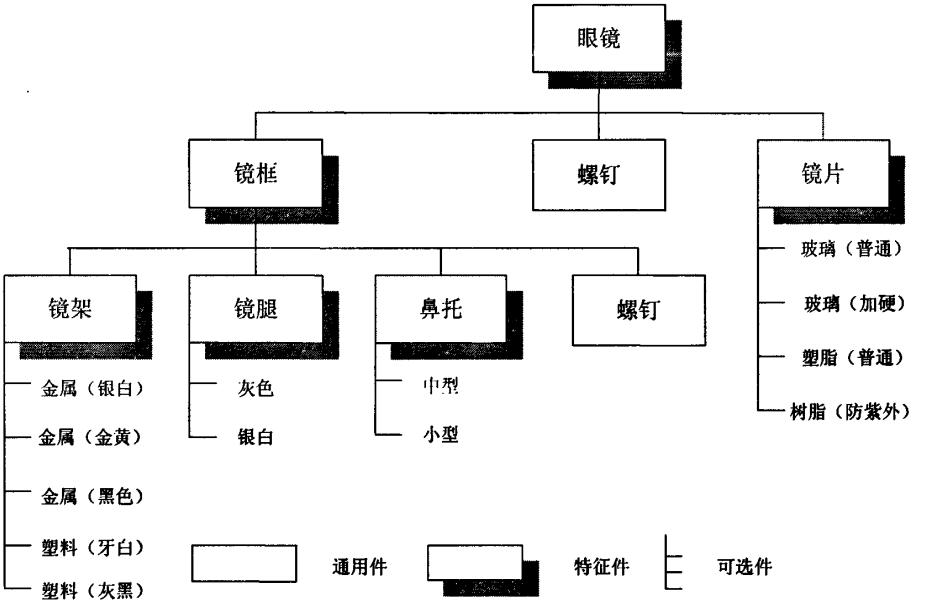

图 2-2 眼镜的模块化产品结构

在模块化产品结构中，往往在多个层次都有选项问题，均可以利用模块化BOM 按照装配最终产品的要求来组建模块。模块化可以得到两个不同的目的：

1) 可以摆脱组合可选产品特征的麻烦。

2) 把通用零件与专用零件区分开来。

2.3 产品族结构树

产品族是共享通用技术并定位于一系列相互关联的市场应用的一组产品。它也是一种利用有限的开发、制造和服务来经济地发展产品多样性的方法[15,16]。产品族的界定有以下三个原则：

- 产品族中的产品具有相同的市场定位和客户群需求
- 产品族中产品具有相似的产品结构，并可用通用结构来表述

- 产品族中产品叶结点上的零部件具有相似的功能和相同的外部接口关系

产品族结构管理的基本功能是建立、维护产品族结构模型，并以此模型为核心来管理工程数据与文档。

2.3.1 基于零部件族管理的产品族结构树

在产品族结构树中，节点被分为两类：概念件和实件[1]。概念件是产品族结构树中设置的一种物理上并不存在的节点，并不代表具体的零部件。与概念件相对应的是实件，实件是真正存在的零部件。概念件又分为设计虚件和装配虚件。

产品族结构树中存在多个设计虚件，设计虚件是具有某些共同特性的实件的一种抽象，这些实件的共性可用参数(属性)来描述，设计虚件对应的这些实件的参数值各不同，所有实件的参数值是一定的，可看成是设计虚件的实例。例如，可以将主板定义为一个设计虚件，其参数为型号、规格等，如果一个产品结构树中使用了该主板设计虚件，表明由该产品结构树描述的所有具体产品可能都需要有主板，至于具体选什么样的主板则需要在产品配置过程中根据客户需求决定，确定其参数值。也就是说，在生成具体产品结构的过程中，配置规则的作用或用户交互选择设计虚件对应的实件的标准，就是基于设计虚件的这些参数的。如果一个零部件是设计虚件，则由它组装成的祖先部件也一定是设计虚件，它们的参数之间存在着某些约束关系，至于是什么约束关系我们并不表示在产品族结构树中。

实际上，设计虚件可以被看成是由多个实件组成的一个零部件类，这些零部件类可以通过零部件族进行管理。零部件族管理(Part Family Management)是建立在编码服务基础上的一种管理功能，它利用成组技术(Group Technology，GT)，根据零部件的基本属性来对零部件进行分类，从而有利于用户快速、高效的查询产品数据，增加零部件和信息的重利用程度。零部件属性分为基本属性和自定义属性。基本属性是指所有零部件都具有的属性，如编码、图号、名称、规格、单位等；自定义属性是指某一类零部件特有的属性，如齿轮类零部件具有的齿顶圆直径、齿数、模数等。一般地，一个零部件族与某个自定义属性集建立着关联。

这里，按照企业零部件的生产方式，将零部件分成自制件和非自制件两个大类，然后对这两个大类进行分类细化，最终形成企业全部零部件层次结构的零部件族结构树。这种树状的零部件族结构能够清晰明确的显示给用户，方便用户对零部件各种信息的查询。

装配虚件即通常所说的虚拟件，装配虚件一般有两个方面的用途。一是作为一般性业务管理使用，其作用只是为了达到一定的管理目的，如组合采购、组合存储、组合发料，这样在处理业务时，用计算机查询只需要对装配虚件操作，就可以自动生成实际的业务单据。二是简化产品结构的管理，如图 3-4 所示，在产品 A 引入装配虚件 E 之前，部件 B 与 D 的定义过程会重复引用到零件 F、G、H，加大 BOM 设计人员的工作量，并且数据库的存储空间也会增加。而引入装配虚件 E 后，在 B 与 D 的定义过程中只需加入一个子件 E，无需重复加入零件 F、G、H，从而达到简化 BOM 结构的目的。另外，可以将 E 作为一个模块化部件对待，从而降低 BOM 在数据库中的存储量。如果在多个产品结构中有大量的相同子件重复出现，这种定义方式的优越性就更加明显。

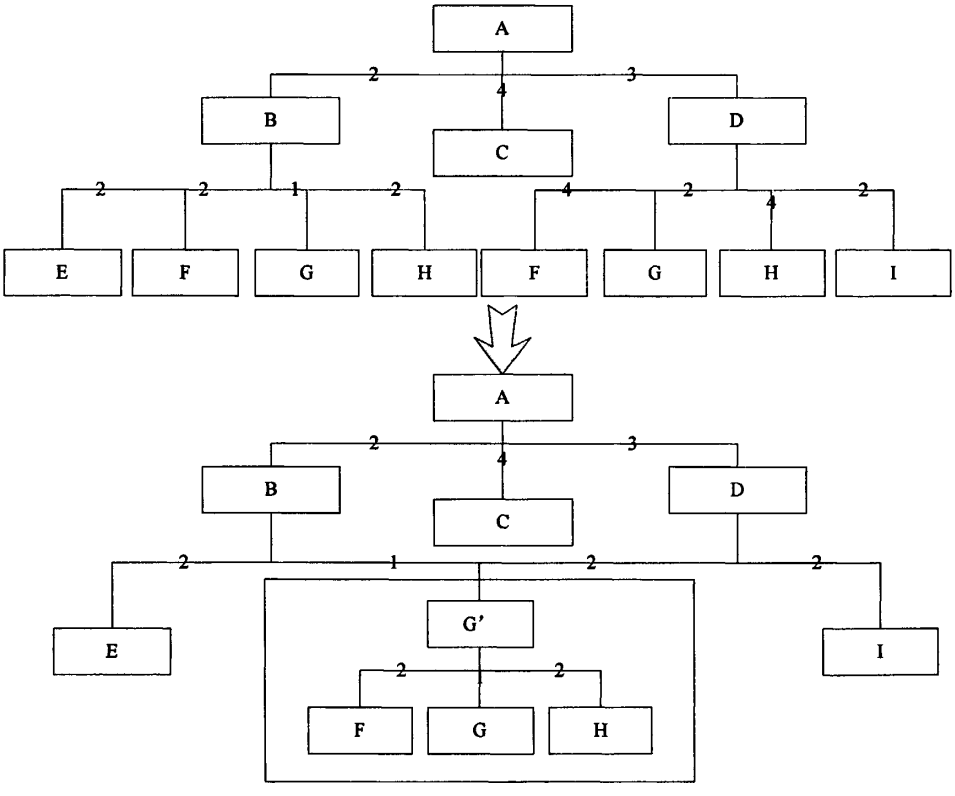

图 2-3 装配虚拟件作用示意图

总之，设计虚件和装配虚件分解了产品系列数据的复杂性，减少了零部件之间的影响，提高了 BOM 结构的适应性，使产品 BOM 的组织更简洁，易于控制和修改。

此外，为了体现产品族结构树局部结构的可改变性，在产品族结构树中，子件相对于父件存在是否选用的约束关系，即产品族结构树中的每一个节点又可分为必选件和可选件，如果一个节点是必选件，则其相对于其父件是必选的，如果一个节点是可选件，则其相对于其父件是可选的。一个部件的所有子件不可能全是可选件，或者说一个部件至少有一个子件是必选件。

综上所述，本文所讨论的产品族结构树出三种类型的物料组成，实件、设计虚件和装配虚件。同时，这三种类型的物料相对于各自的父件来说又存在是否选用的约束关系。设计虚件体现了产品族结构树总体属性可配置的动态性，可选件相对于父件存在的是否选用的约束关系体现了产品族结构树局部结构可配置的动态性。

2.3.2 产品族结构树定义

前面的章节已提到，BOM 的基本结构是一个带权有向无环图，可表示为一个三元组(M，E，f)，其中 M 是图中的节点集，即有限物料的集合；$E \subset M \times M$，E 表示图中的有向边的集合，即父子关系的集合；f 是从边集合 E 到实数集合的函数，$\forall e \in E$，f(e)是 e 的权，表示在父子关系中，子件组成父件的用量[14]。

另外满足如下约束：

1、$\forall m \in M$，$|\{r \in M | <m, r> \notin E\}|=1$；

2、$\forall m \in M$，$<m, m> \notin E$；

3、$\forall m \in M$，$<m, m> \notin E^+$，其中 E^+是指集合 E 的传递闭包。

现在根据上面提到的产品族结构树相关概念的介绍，结合 BOM 的基本结构的定义，给出本文所讨论的产品族结构树(设计 BOM 结构)的形式化定义。

产品族结构树是这样一个有向图，可表示为如下七元组：$dBOM_i=(F_i, V_i, S_i, E_i, f_{Di}, f_{Ci}, f_{Fi})$，其中：

F_i 是 $dBOM_i$ 中设计虚件的集合；

V_i 是装配虚件的集合；

S_i 是实件的集合；

E_i 是序偶集，其中序偶的元素来自 $F_i \cup V_i \cup S_i$；

f_{Di} 是从 E_i 到实数集合的函数；

f_{Ci} 从 E_i 到{main，option}的函数，其中 main 和 option 分别表示必选约束和可选约束；

f_{Fi} 是 F_i 到 P(Material)的函数，其中，Material 为实件的全集，即企业中所有具体产品的产品结构树中存在的物料的集合，P(X)表示 X 的幂集。

另外满足如下约束：

1、Fi、Vi、Si 两两相交均为$\varnothing$；任何一节点只能是设计虚件、装配虚件、实件中的一种；

2、$\forall m \in Fi \cup Vi \cup Si$，$|\{ri \in Fi \cup Vi \cup Si \mid <m, ri> \notin Ei\}|=1$；有且仅有一个入度为零的节点 ri 即根节点，代表用户感兴趣的对象，可以是产品也可以是部件；

3、$\forall m \in Fi \cup Vi \cup Si$，$<m, m> \notin Ei$；每个节点不允许自己和自己构成父子关系；

4、$\forall m \in Fi \cup Vi \cup Si$，$<m, m> \notin Ei+$，其中 Ei+是指集合 Ei 的传递闭包；每个节点不可能是自身子件的子件；

5、$\forall m \in Fi$，$m \neq ri$，如果$<m0, m> \in Ei+$，则 $m0 \in Fi \cup Vi$，并且$\exists m1 \in Fi$，$<m1, m> \in Ei+$；如果一个非根节点是设计虚件，则其所在路径上的所有祖先节点均为设计虚件或装配虚件，且至少存在一个祖先节点是设计虚件；

6、$\forall m \in Fi \cup Vi \cup Si$，$|\{m0 \in Fi \cup Vi \cup Si \mid <m, m0> \in Ei\}| \neq 0$，$\exists m1 \in Fi \cup Vi \cup Si$，有$<m, m1> \in Ei \wedge fCi(<m, m1>)$=main。一个非叶子节点

的所有子件不会全是可选件，或者说一个非叶子节点至少有一个子件是必选件。

dBOMi 可用图形的表示方法表示如图 2-4，图中表示的是计算机产品系列的结构，其中省略了父子件的装配数量关系，该产品族结构树示例是满足上述的约束定义的。

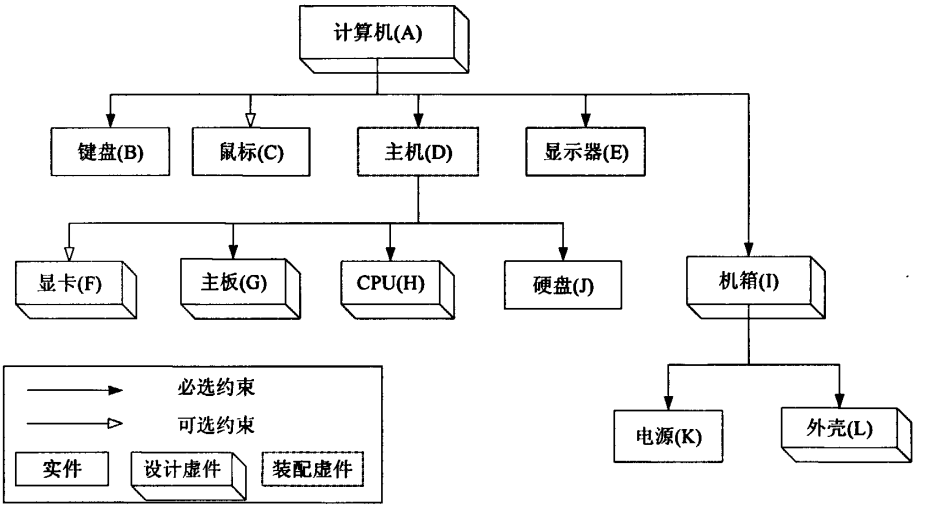

图 2-4　产品族结构树图示

为了方便讨论，我们假设属于该产品系列的所有计算机都是使用同一种键盘、鼠标、显示器、硬盘和电源，即为实件，其他零部件均为设计虚件，用零部件族进行管理，每个设计虚件均含有一定的参数，用文献[35]中定义的语言对图 2-4 中的产品族结构树中设计虚件对应的参数描述如下：

```
/**
**计算机产品族的结构描述
**/
component A is
      name(computer);
      series(高档|中档);
      color(white| black| silvery);
end component;
body A is
      include B;
      include C;
      include D;
      include E;
      include I;
end body;
component B is
```

```
    name(keyboard);
    spec(abc);
end component;
component C is
    name(mouse);
    spec(abc);
end component;
component D is
    name(mainframe);
end component;
body D is
    include F;
    include G;
    include H;
    include J;
end body;
component E is
end component;
body E is
    name(display);
    spec(samsung550s);
end body;
component F is
    name(display card);
    spec(S128| GF256);
end component;
component G is
    name(mainboard);
    spec(intel615| intel915|amd15);
end component;
component H is
    name(cpu);
    spec(intel1.2| intel2.0| duron800);
end component;
component I is
    name(computer box);
    color(white| black| silvery);
end component;
body I is
    include K;
    include L;
end body;
component J is
    name(harddisk);
```

```
    spec(scsi-unit);
end component;
component K is
    name(power supply);
    spec(p-11);
end component;
component L is
    name(shell);
    color(white| black| silvery);
end component;
```

2.4 BOM 模型的遍历算法

BOM 模型是由双亲件和子件构成的关系树，当将产品结构的数据输入计算机，并以单级 BOM 的数据表形式保存后，就可以对 BOM 进行自顶向下的分解或是自底向上的跟踪操作，并能根据不同用户的需求以不同的格式显示出来，以满足企业中各种用户的需求。

由于父子物料的相对性，分解算法与跟踪算法从本质上说是类似的。因此，本节的以下部分将从 BOM 分解的角度来对 BOM 遍历算法进行介绍。典型的 BOM 遍历算法通常包括递归算法和分层算法。

2.4.1 BOM 遍历的递归算法

递归算法采用了先根遍历的方式，它以产品结构树中的节点是否有子节点为递归条件，充分展现了物料间的层次关系和 BOM 树的结构。

其算法流程如下[18]：

(1) 找出 BOM 数据表中“父物料编号”为空的子项即为产品项，记为结构树的根节点，将其对应记录的“访问标记”项置 1。

(2) 找出一个以产品项为父项的子项，获取其层次码，将其对应记录的“访问标记”项置 1。

(3) 以新节点为父项找出一个子项，将对应记录的“访问标记”项置 1，并以其为父项重复递归查找直到最下层没有子项为止，返回上一层节点递归下一子

项，直到返回到第二层节点。

(4) 查找出以产品项为父项的另一节点，将其对应记录的“访问标记”项置 1，以它为新的父项重复步骤(3)。

(5) 重复步骤(2)、(3)、(4)直到所有记录的“访问标记”都为 1，此时表明所有物料都已经访问过，遍历结束。

以图 2-1 所示的产品结构为例，其递归遍历次序为：眼镜、镜框(1)、镜架(1)、镜腿(2)、鼻托(2)、螺钉(4)、镜片(2)、螺钉(2)。该算法的优点是较好的展现了 BOM 的结构和层次关系，并且程序简单；缺点是由于使用到了递归，涉及到了堆栈的操作，在进行深层次遍历时，系统消耗的资源会很大。

2.4.2 BOM 遍历的分层算法

分层算法采用了层次遍历的方式，它首先访问层次数为 0 的根节点，然后访问层次数为 1 的所有节点，依次逐层往下遍历直到访问完最后一层的所有节点。其算法流程如下：

(1) 找出 BOM 数据表中“父物料编号”为空的子项即为产品项，记为结构树的根节点，将其加入队列。

(2) 判断队列是否为空，如果空则遍历结束，此时表明所有物料都已经访问过；否则进入步骤(3)执行。

(3) 获取队首元素的层次码，并将其出队列。获取该记录的子节点，并将其入队列。跳到步骤(2)执行。

以图 2-1 所示的产品结构为例，其层次遍历次序为：眼镜、镜框(1)、镜片(2)、螺钉(2)、镜架(1)、镜腿(2)、鼻托(2)、螺钉(4)。该算法能在一次搜索过程中将节点的子件全部搜索出来，大大地提高了搜索的速度；但是由于搜索时分层进行的，物料的存放顺序未能展现物料间的父子关系，而在进行物料结构的取代、删除时，需要知道物料间的父子关系，因而 BOM 遍历的分层算法将无法进行这些操作。

第3章 基于面向对象技术的动态 BOM 模型

通过对前面所讨论模型的研究可以发现，它们存在着以下方面的不足：这些模型中的产品部件之间有着很强的依赖性，不利于进行动态的维护，也不能对客户的需求做出快速的反应。另一方面，对于不同的制造企业来说，所使用的 BOM 的种类、形式、意义和用途都可能是不一样的。而在每个制造企业内部，也有可能使用多种具有不同意义的 BOM。对 BOM 进行动态建模[19,25]，使得每个企业可以根据自己的具体生产制造情况建立不同的 BOM 对象模型，或者在企业运行过程中根据需要改变已有的 BOM 对象模型，无疑具有重要的意义。

3.1 动态建模思想

随着面向对象思想的发展，对象模型得到了研究人员极大的重视，并被广泛地应用。虽然关于对象模型[34,38,41]还没有一个统一的定义，但是可以认为对象模型是关于对象和对象间关系的描述。因此，建立对象模型就是描述一系列的实体对象以及描述这些实体对象之间存在的关联、继承和聚合关系。当关联关系比较复杂时，可以通过建立关系对象来进行描述。

为了实现对动态 BOM 对象模型的定义，必须采用动态建模思想，从更为抽象的层次上来描述 BOM 的对象模型。BOM 对象模型需要具有充分地描述对象类以及对象类之间关系的功能。通过该对象模型实现对象类的可定制化，在企业需求的约束下，生成一系列适合企业具体情况的类，组成 BOM 的对象模型。

3.2 BOM 数据组成分析

从 BOM 视图的观点看，BOM 主要体现了从某一角度看到的产品结构和该产品结构下对象的属性。所有 BOM 的并集构成了完整的产品结构。BOM 所包含的信息主要有两类，一类是产品结构信息，另一类是产品结构中对象的信息。从面向对象的观点出发，整个企业的运行可以看作是通过许多类型的对象的相互交互完成的。我们可以把这两种信息用两个类来描述：实体类和关系类。实体类用于描述抽象出的企业具体的业务实体类。这些业务实体类可以是零件、部件、

工序等。由于 BOM 是从某一角度来描述产品结构，因此，每一个 BOM 所观察到的产品结构树中的对象的属性可能相同也可能不同。相应地，将实体类的属性分为两类：一类是基本属性，它属于所有的 BOM 视图，在任何 BOM 视图中都是可见的；另一类是特征属性，这类属性仅与某一个或几个具体的 BOM 类型相关联，称之为产品数据与 BOM 之间的第一维相关性。例如，零件的生产提前期属性是属于 MBOM 视图的，在 EBOM 中是不可见的，所以这一属性是关于 MBOM 的特征属性。这类属性在动态建模描述时是要指定其对应的 BOM 类型的。关系类用于描述两个实体类之间的结构关系，如部件与零件之间的包含关系。它可以有自身的关联属性。不同的关系对象是与不同 BOM 相关联的，例如部件 A 与外协件 B 之间的结构关系可能存在于 EBOM 中但不存在于 MBOM 中，这种结构关系是与 EBOM 相关联的，称之为产品数据与 BOM 之间的第二维相关性。综上所述，关系类是一个复杂的组合类，它的属性域不仅包含自身的基本属性，还包括了实体类；实体类又是由基本属性和特殊属性组成。

3.3 面向对象模型

BOM 的面向对象模型用于动态的建立 BOM 对象模型。下面我们分别给出 BOM 的形式化定义和元模型定义。

3.3.1 模型的形式化定义

为了指导具体的建模，我们首先给出 BOM 模型的形式化定义：

定义 1　OOBOM = (M，　R，　OP)

其中，OOBOM 代表面向对象的 BOM 模型，M 表示构成 BOM 的物料集合，由于一个产品的 BOM 是树状结构，所以 M 中的每个元素表示树状结构中一个节点，称为物料节点；R 表示物料节点之间的构成关系的集合；OP 表示对 BOM 操作的集合。

进一步对定义 1 中的三个集合作如下的定义：

定义 2　$M = \{(m_1, m_2, \ldots, m_n) \mid m_n = ProM \mid CompM \mid LeafM \mid OptM,$

$0 \le i \le n$，$n \ge 0$}

其中，ProM 表示产品，它往往是 BOM 树的根节点；CompM 表示中间节点，通常是产品的构成部件；LeafM 表示叶子节点，指产品中标准件、外购件或原材料等；OptM 表示可选择节点，指产品中的可替代部件、BOM 中的虚项或带有可选择参数的部件等。

定义 3　R = {($r_{i,j}$<N>) | $r_{i,j}$= IsSub<N> | IsOpt <N> ，N≥1，$0 \le i \le n$，$0 \le j \le n$}。

其中，　$r_{i,j}$< N >表示 BOM 树中两个节点 m_i 和 m_j 之间的父子及构成关系，即 m_i 是 m_j 的父节点，并且 m_i 需要 N 个 m_j。IsSub<N>表示关系中的简单装配关系；IsOpt <N>表示关系中子节点是 OptM 类型。

定义 4　OP = {(op_1，op_2，…，op_n) | op_i in {Add，Delete，Update，Search，etc.}，$0 \le i \le n$}

其中，　op_i 表示一个可以对 BOM 进行的操作，如查找、删除、增加、汇总等。

3.3.2　元模型定义

BOM 对象模型是由实体类和关系类组成的，因此，在 BOM 对象模型中，需要建立一系列类来描述实体类和关系类。由于这些类是用来生成其他类的，因此称其为元类。根据对 BOM 的数据组成的分析，BOM 对象模型主要包括 6 个元类，通过这 6 个元类来进行 BOM 对象模型的描述。

图 3-3 展示了 BOM 对象模型，下面给出这个模型的定义。

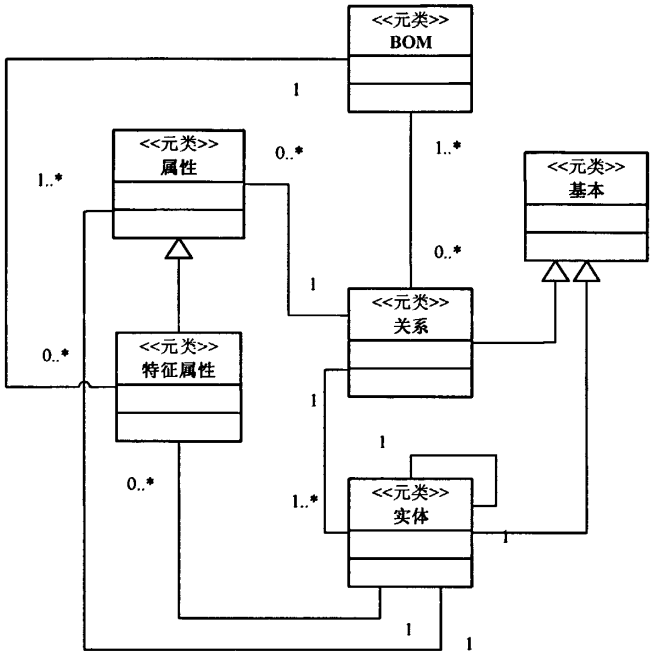

图 3-4 BOM 对象模型

下面对图 3-4 BOM 对象模型中的元类做具体的介绍：

(1)实体元类

实体元类用于描述企业中的实体，它用来规定 BOM 模型中所包含的实体类的数量。实体类可以有产品类、零件类、部件类和文件类等。如果能对实体元类进行较好的设计，则可以利用继承机制来实现代码的重用，也有利于保证数据库基本结构的稳定性。

(2)关系元类

关系元类用于描述实体类之间存在的结构关系，如产品类与组成它的直接子类之间的关系，企业可以根据自身的具体情况设计关系元类。通过关系类型的定义，约束了实体类之间的结构关系。由图 3-3 的 BOM 对象模型可以看出，任一种实体类和关系类既可以只属于某一种 BOM，也可以属于多种 BOM。如部件类与零件类之间的结构关系可能既存在于 EBOM 中又存在于 MBOM 中，但零件类与工序类之间的结构关系可能只存在于 PBOM 中。一种 BOM 中到底包括哪些关系类是由 BOM 元类决定的。

(3)属性元类

属性元类用于描述实体类的基本属性和关系类的关联属性。对于每一个已定义的实体类型，可以建立它的基本属性。例如对产品类建立产品名称、产品描述等属性。对于每一个已定义的关系类，属性元类可以建立其关联属性。如描述产品类和部件类的对象关系类中可以有关联属性“装配数量”、“装配说明”等。这些属性是经过筛选而得到的关键属性，在每一个实体类或者关系类中都是必不可少的，因而可复用性很强。

(4)特征属性元类

特征属性元类用于描述实体对象在各种 BOM 中的特殊属性。在定义时，要考虑特殊属性与各种 BOM 之间的关联关系；并且，由于同一对象的特征属性之间可能存在某种函数映射关系，因此，还要注意在不同特征属性之间建立函数映射关系。

(5)BOM 元类

BOM 元类用于描述 BOM 的具体对象模型的总体情况如 BOM 的名称等，以及所包含的关系类型。前面 4 个元类定义了企业运行时需要的完整的实体类(包括其属性)和关系类(包括其属性)，BOM 元类通过选择不同的关系类来定义 BOM 的总体结构，关系类中定义了所包含的实体类和针对某 BOM 的约束规则，实体类中定义了该类的属性和针对某 BOM 的特征属性。

从这个模型中可以得出 BOM 的一个一般类(称为 BOM 元类)，所有不同类型产品的 BOM 都作为一个具体 BOM 类来处理，并且都从这个类进行继承，一个具体产品的 BOM 实例可以由它所在的 BOM 类来创建。有了这个模型，就可以建立各种 BOM 之间的联系。例如，设计部门的 BOM 可以从这个一般的 BOM 继承，而工艺部门的 BOM 又可以从设计部门的 BOM 继承，只不过为 BOM 添加了自制件的工艺信息；成本 BOM 也可以从设计部门的 BOM 中继承而来，并添加了成本信息。对于系列产品的 BOM 或改进产品的 BOM 都可以用继承的方法建立它们之间的联系。从 BOM 基类继承来的所有类及其继承关系构成了 BOM 类层次。这个模型为产品 BOM 数据的建立、维护和管理带来很大的方便。

第4章 面向对象 BOM 模型的设计与实现

为了对该模型设计的合理性以及运行的效率进行检验，本文给出了一个简单的系统以便于对其进行分析和比较。

4.1 系统介绍

我们要实现一个客户订制眼镜的系统，注册用户可以在网上登陆，然后可以根据自己的需要定制、取消和查询自己所定制的眼镜。系统需要根据用户的定制信息分析、获取建立 BOM 结构所需要的要素。对这个简单系统的要求是能快速、准确的分析客户需求并建立起该产品的 BOM 表；在客户进行定制查询时，也能有较快的反应速度。

4.2 系统的总体设计和体系结构

4.2.1 系统的总体设计

根据系统特点和设计要求，系统的总体设计如下：

1) 分层设计。我们采用了 B/S 架构，根据 J2EE 的多层体系结构，我们把系统层次细化：分别为：Web 层(表示层)、业务逻辑层、持久化层和数据库层。每一层都是逻辑独立的，并且实现相对独立的功能。层与层之间分工明确而又相互协作。

2) 采用 Struts 框架结构。Struts 框架具有组件的模块化，灵活性和重用性的优点，同时简化了基于 MVC 的 Web 应用程序的开发。它减弱了业务逻辑接口和数据接口之间的耦合，让视图层更加富于变化性。此外，它的页面导航能力更是突出。

3) 采用 Hibernate 持久化对象。在传统的 J2EE 设计中持久层采用的是 EntityBean，但是采用 Hibernate 作为持久层管理技术会有更高的效率。Hibernate 的优点在后面的章节进行介绍。

4) 工程化的开发。这个系统采用的是一个工程化的开发过程。从开发工具的选择到模块化设计，我们都从软件工程的角度出发，综合的进行了考虑。系统的可扩展性，可维护性，开发效率都是系统价值的重要指标。按照上面的叙述，系统的整体设计如下图 4-1 所示：

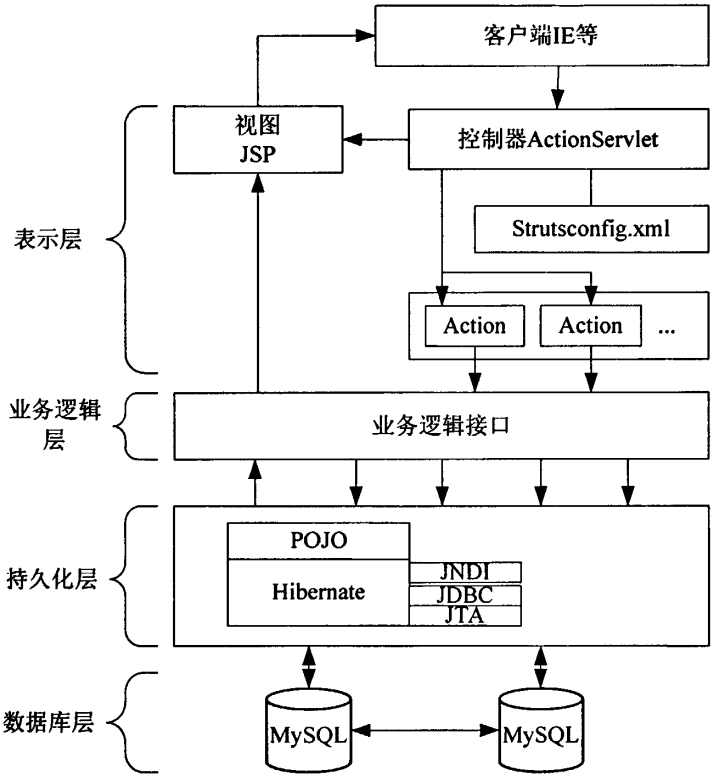

图 4-1 系统设计结构图

4.2.2 系统的体系结构

J2EE 应用程序的设计都是与 MVC 的设计模式相结合的。M(Model)表示模型，处理核心业务实体及其相关的业务逻辑，模型对象使用一组稳定的可复用的业务对象来实现，开发人员在多种应用程序中使用这些对象。V(View)表示视图，处理应用程序的数据表示和对用户的命令，视图对象可以是 HTML 网页，也可以是 JavaApplet 客户程序。C(Controller)表示控制器，位于视图层和模型层之间，实现工作流，视图层的用户发出的命令触发控制层中的代码，这些代码操纵模型层中的一个或多个对象来完成命令功能。MVC 的设计模型根据功能清楚的分开

了代码，实现了逻辑上的独立，降低了各层之间的耦合度，提高了开发团队的效率。

在本系统中，用户界面可以看作是 V，它是由浏览器显示给用户的界面；业务逻辑层和持久层则为 M，它是系统的核心部分，由 POJO 和 Hibernate 对象组成；Web 层对应 C，它控制用户对业务逻辑的操作，同时控制把操作结果显示给用户，由 Servlet 程序实现。系统的 MVC 模型图如图 4-2 所示，其中 POJO(Plain Old Java Object)是简单的 Java 对象，实际上就是普通的 JavaBean，这个对象所传递的数据通过 JSP 页面显示给用户。

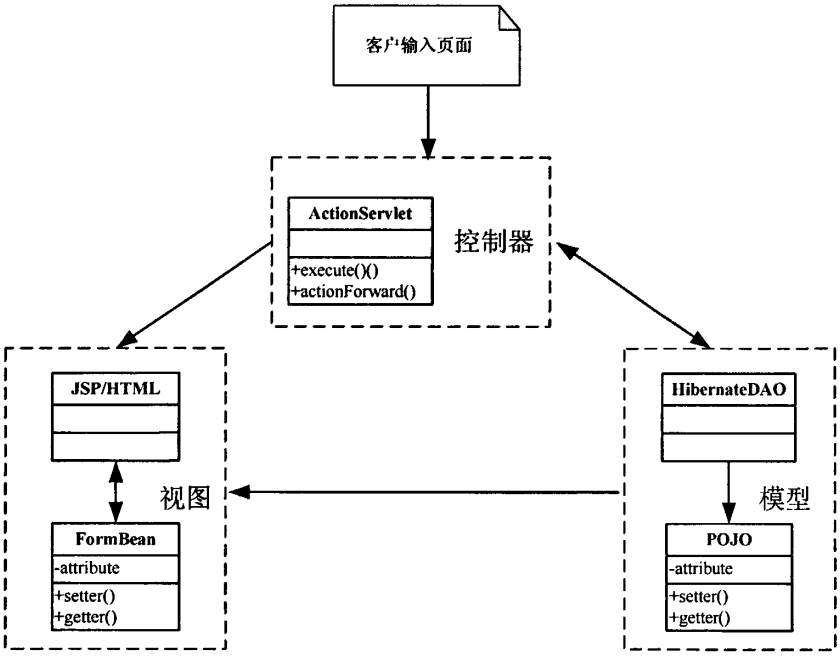

图 4-2 系统的 MVC 模型图

4.3 系统中的关键层次设计

4.3.1 对象关系数据库映射

对象模型能够更好的抽象和描述现实世界，但对象数据库的发展却还并不成熟。对于实际应用中需要处理的数据，关系数据库仍是人们青睐的持久信息存储工具。因此，在构造 BOM 模型时，我们需要考虑对象到关系数据库的映射问题。

4.3.1.1 ORM 技术

面向对象设计机制遵循的软件工程原理如封装、耦合、关联、多态等，侧重于是用包含数据和行为的对象来构建应用程序，而关系模型基于关系数学理论。两者之间的这种“阻抗不匹配”[3]使得把面向对象的一些操作映射到关系数据库非常麻烦，需要大量的重复代码，并且经常需要修改业务层的代码和 SQL 语句来适应处在数据层的数据库表的变化。为了解决上述问题，我们使用对象关系映射(ORM，Object-Relational Mapping)技术[3]。

ORM 的引入，即在业务层和数据层之间添加一个数据访问层，封装与数据库的各种操作。这里我们用到了一个对象关系数据库应用模型：在业务逻辑层和关系数据库的物理存储结构中间增加一个 O-R 代理层。代理层为用户应用程序提供统一接口，并结合 RDBMS 透明地完成对象和关系数据库之间的转换，解决对象数据的存储和查询等操作问题。下图 3-1 为系统的结构图

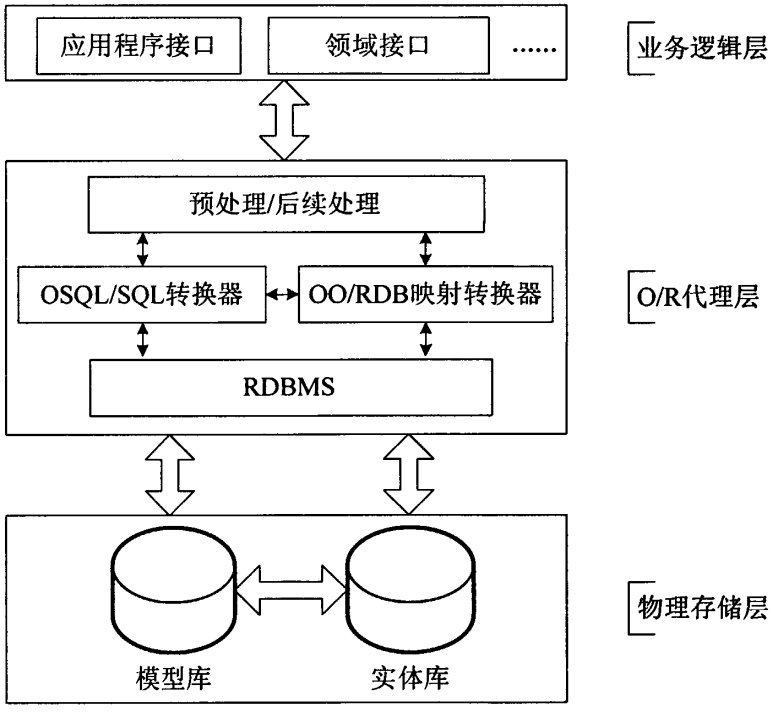

图 3-1 系统结构图

4.3.1.2 业务逻辑层

业务逻辑层是开发人员和用户所接触的接口层，在这个层次上是完全面向对象的内容。业务逻辑层由很多接口组成。例如，应用程序接口是专门为特定的应用程序开发的面向对象接口；领域接口针对具体的应用领域(垂直应用)，如某些领域接口仅适用于特定的某一个领域。业务逻辑层还可以包括其他类型的接口，例如多领域接口表明他们适用于多个领域。

4.3.1.3 O/R 代理层

O/R 代理层是系统的关键层和核心层。它是与领域无关的代理层，用于完成 OO 和 RDB 的相互映射和转换。

(1) 预处理/后续处理模块：在映射转换的前期和后期，做一些预备性或结尾性工作，如面向对象的各类概念表示及其有效性维护和柔性扩展等。

(2) OSQL/SQL 转换器：负责将 ODDB 语言(OSQL)转换成语义等价的 RDB 语言 (SQL) 。

(3) OO/RDB 映射转换器：从数据库语言的角度，转换分为两类：从 OODB 到 RDB 的数据操纵语言(DML)的转换及其数据描述语言(DDL)的转换。OO/ RDB 映射转换器和 0SQL/SQL 转换器共同配合完成这两项工作，但侧重有所不同。OSQL/SQL 转换器则主要侧重于数据操纵语言之间的转换；OO/ RDB 映射转换器主要侧重于描述语言之间的转换，即主要负责面向对象数据模式和关系数据模式的映射。该映射保证了对象的语义在相应的关系中不会丢失，并为数据操纵语言之间的转换奠定基础。

(4) RDBMS：成熟的关系数据库管理系统。O/R 代理层最终通过这个模块完成对底层关系数据库的存储和各种其它操作。

4.3.1.4 物理存储层

系统的最底层是物理存储层，在数据存储的结构上仍然采用关系数据的形式。该模型是对传统关系数据库的扩展，是把面向对象技术和关系型数据库相结

合而形成的关系对象数据库模型。

4.3.2 BOM 的数据库设计

面向对象数据模型是新的数据表达模型，它允许根据复杂对象，而不是记录和关系进行应用设计。从面向对象的观点看，类相对于类型，而对象相对于实例。通过发送消息到对象，对象被激活后，其所定义的方法就可以对对象数据进行操作。我们首先抽象出 BOM 模型所包含的类，每一个类对应关系数据库中的一张表，表中的每一列对应类中的属性。这样，我们就可以充分利用成熟的面向对象技术进行业务逻辑方面的操作，用对象关系映射(ORM，Object－Relation Mapping)机制来完成类对关系表的管理。

4.3.2.1 数据库结构

BOM 底层数据结构对整个方案的成败具有重要的影响。BOM 的数据主要包括两个部分：一部分是关于建模信息的，另一部分是关于实体对象和关系对象信息的。可以在数据库中建立两个模式：一个用于保存各种建模信息，称为模型库；另一个用于保存企业运行中的各种实体对象和关系对象，称为实体库。BOM 的信息在建模工具的支持下保存到模型库中，成为模型库要保存的重要内容。然后，再根据模型库的建模信息修改实体库的逻辑结构，即建立保存实体对象信息和关系对象信息的一系列表，同时实体类和关系类之间的逻辑关系仍然保存在模型库中，并不映射到表与表之间建立的引用关系。图 3-5 显示了系统动态建模时数据的流动。模型库由于保存了具体企业的 BOM 对象模型，在主程序运行时起到了极其重要的作用。它相当于数据字典，主程序对实体库任何对象的访问要通过查询模型库的信息获得关于该对象所属类的基本信息。

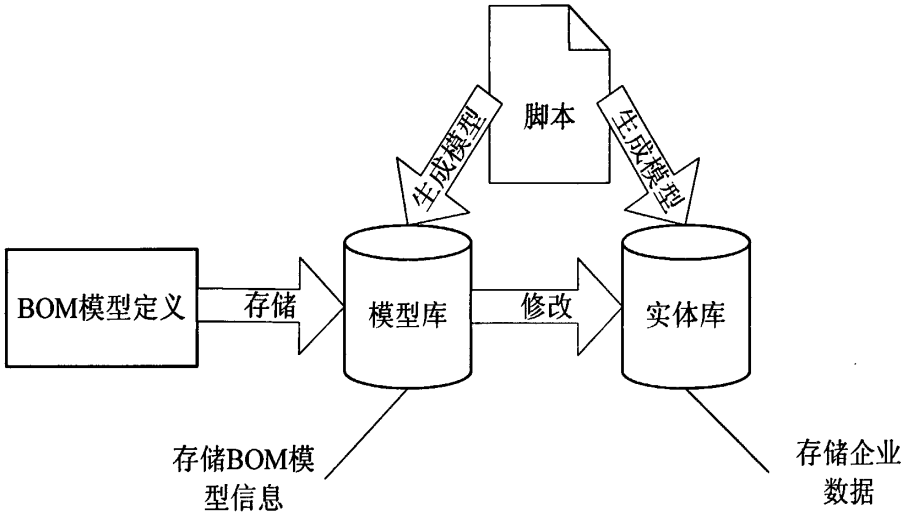

图 3-5 系统动态建模数据流动图

4.3.2.2 对象属性到关系属性的映射

把 BOM 对象的属性映射到关系的属性，则一个 BOM 对象的属性就表示为关系中的一个元组，在关系数据库中就成为了表中的记录。映射时要注意以下的问题：

(1)如果 BOM 对象中的一些属性可以经过其他属性计算而得到，这种属性就可以不进行映射。

(2)如果 BOM 对象的属性是一个复杂的数据类型，如其他的对象，那么在把这种属性映射到具有原自行要求的数据库中的关系属性时，要进行相应的处理，可以用多个关系属性来表示一个对象属性，也可以用另外一个关系来实现一个对象属性，这种情况可通过相关属性来建立联系，也可利用外键等工具来保持数据的一致性和完整性，所以对象属性到关系属性的映射不是一对一的。例如，一个眼镜是由镜框，镜片等组成的，而镜框又是由镜架、等组成的复杂对象。这样，我们就可以将镜框这种属性映射为一个镜框表，再在鼻托 BOM 的关系表中增加一个对象标识作为与 BOM 关系表建立关联的外键。

另外，在面向对象系统中，每个对象都有唯一的对象标识符(Object Identifier，OID)。而在关系数据库中表示一条记录的属性称为关键字。当对象属性映射到关系数据库是，对象的标识符将被映射为关系数据库中的关键字。在关系数据库

中就使用到了代理主键，下面介绍一下这个概念。

在 Student 表中，我们可以把 No(学号)字段作为主键。No 字段时具有业务含义的字段，把这种主键称为自然主键。尽管也是可行的，但是不能满足不断变化的业务需求，一旦出现了允许学号相同的业务需求，就必须修改数据模型，重新定义表的主键，这给数据库的维护增加了难度。代理主键是不具备业务含义的字段，只是用来唯一的标识一条记录。使用代理主键主要有以下几个优点：

- 在业务发生变化时，适应性强
- 保证系统的一致性和系统可操作性
- 数据表连接和更新的性能更高
- 存在主从关系时，代理主键更新方便
- 在并发环境下，更容易实现唯一的 ID
- 系统占用的存储空间更省

4.3.2.3 继承结构的映射

由于关系数据库不支持继承，这就要求必须将对象模式的继承结构映射到相应的数据库模式中去。类的继承结构到关系数据库的映射一般有三种方法[3]：

- 整个类层次映射成单个表
- 每个具体类都各自映射成一个表
- 每个类映射成一个表

如图 3-6 所示，描述了 3 个类的层次结构：一个抽象类 Component，两个继承类 Part 和 Material。

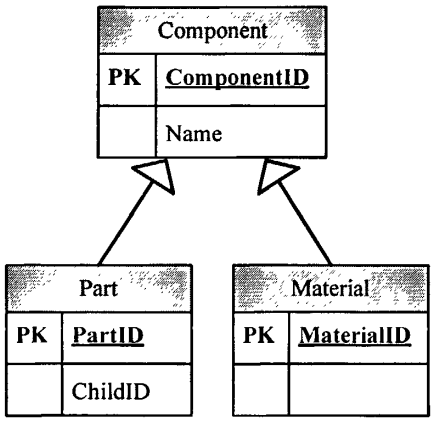

图 3-6 一个简单类层次图

为了简单起见，这里省略了类的许多属性，也没有对类的任何操作进行建模，下面我们来具体看一下三种映射方案。

1、整个类层次映射成单个表

整个类层次映射为单个数据表，即将具有相同父类的所有类都映射到一个数据库表中，这些类属性映射的集合组成这个表的列，在这种策略下，只需要对最底层的类进行映射。

当采用这种策略时，图 3-6 中的类层次结构对应的数据模型如图 3-7 所示。每个类的属性都存储到表 Component 中，表名一般采用类层次结构中根类的名字来命名。

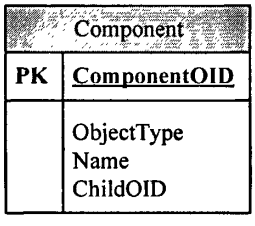

图 3-7 整个类层次映射成一个表

其中，ObjectType 字段用来具体类型，本例中就是指明该组件是一个零件还是物料。ComponentOID 和 ChildOID 字段使用的是一个代理键。

用来识别对象类型的 ObjectType 字段是必需的，这个对象可以由给定的数据库中的一行数据实例化而来。例如，取值为 P 表示该组件是雇员，M 表示是物料。

采用这种策略，优点是比较简单，添加新类很方便，只需要为新加的数据添加新的类型标志即可。因为所有数据都在一张表中，因此数据的访问速度很快，容易生成专门的报表。

由于所有的类直接关联到同一张表，类层次结构内的耦合度增加。修改一个类将影响整张表，而整张表的改变又会影响类层次结构中的其他类。这个表要容纳所有的子类的字段，很多字段只是对某个类有意义，而对其他类则纯粹是多余的，导致数据库空间存在着潜在的浪费。

对于那些类层次结构内部的类型间不存在或存在极少重叠情况的简单类层次结构或继承深度比较前的类层次结构而言，这种策略较好。

2、每个具体类都各自映射成一个表

每个具体类都各自映射成单个表，即为每个具体类创建一张表[35]，每张表既包括对应类自己实现的那些属性，也包括他继承下来的那些属性。

采用这种映射方法，图 3-6 中的类层次结构所对应的数据库物理数据模型如图 3-8 所示。具体类 Part 和 Material 都有对应的映射表，对象从这些表中被实例化，而抽象类 Component 则没有对应的表。分别为每张表分配了对应的主键 PartOID 和 MaterialOID。

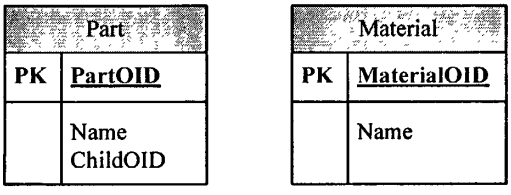

图 3-8 每个具体类都各自映射成一个表

在这种策略下，由于单个类的所有数据都存储在一张表中，所以特别容易生成专门的报表，访问单个对象数据的性能高，而且没有数据冗余。缺点是类修改的时候必须修改它对应的表，同时也必须修改它所有字类对应的表，而且在搜索的时候比较麻烦。例如，当搜索符合某个条件的 Component 对象时，需要同时搜索 Part 表和 Material 表；并且，当在继承树中新增一个继承节点的时候，需要新增一个表，搜索的范围也必须扩大。

每个具体子类映射成单个表，这种策略适用于那种类型很少改变，类型之间

极少重叠的情况。

3、每个类映射成一个表

每个类均映射成为数据表，即每个类的属性(不包括从父类继承来的非主键属性)映射组成这个表的列，在这这种策略下，居于继承关系的类被映射到不同的表中，表之间通过主键关联。

将每个类都映射成一张单独的表，图 3-6 中的类层次结构对应的物理数据模型如图 3-10 所示。类 Part 的数据被存储在两张表 Component 和 Part 表中，因此要获取这些数据，需要连接着两张表(或者分两次读取，每张表读一次)。

这种策略下，键的应用非常重要。注意 ComponentOID 是如何作为所有表的主键来使用的。对于 Part 和 Material 表来说，ComponentOID 既是主键又是外键。对于 Part 表，ComponentOID 是它的主键，同时也作为外键来维系与表 Component 之间的关联。

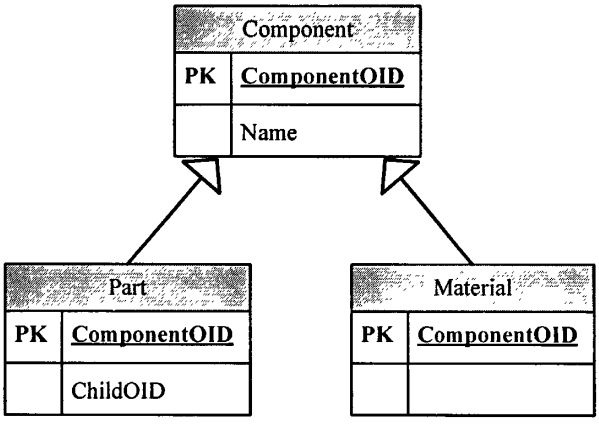

图 3-10 每个类映射成一个表

这种策略很容易理解，因为和类图非常相像，每种类型的记录分别在相应的表中，因此可以很好的支持多态。修改父类或添加新的子类时，只需要简单的修改或添加一张表。

每个类一张表，则在数据库中会有很多表(还需要附加表用来维护类之间的关系)。使用这种方法，使得读写数据需要更长的时间，应为需要访问多张表。从数据库生成专门的报表特别困难。

在类型之间有较大重叠，或类型会频繁的修改的情况下是和使用这种策略。

下面让我们来比较一下这几种映射策略的优缺点。

如表 3-1 所示，继承映射策略而言，没有一种策略能够满足所有场合。比较适合的方案是：先用整个层次结构映射一张表的策略，接下来如果需要，就重构相应的模式。

表 3-1 继承映射策略的比较

考虑因素	整个类层次一个表	每个具体类一个表	每个类一个表
实现	容易	较易	难
数据访问	容易	较易	较难
数据访问速度	快	快	较快
对多态性的支持	中	高	低
对类图的扩展支持	较易	较难	难
耦合程度	非常高	高	低

这三种方法各有优缺点。根据本文设计的特点，我们在具体应用时就采用方法 1。要把 BOM 类层次中所有 BOM 类存储在一个表中，只要把 BOM 类层次设计的所有属性映射到这个表中即可。为了更好的保存和描述继承关系，系统中要设置一个存放基类和派生类的继承关系表，当基类数据有变化时可以通过数据库的触发器来维护派生类数据的一致性。由于关系类描述的对象关系都被约束在确定的两种类型之间，因此，BOM 的结构关系数据是单层存储的。这种方式数据冗余少，同时由于单层 BOM 相同的关系对象只记录一次，因此 BOM 表更改简单。

4.3.2.4 对象间关系的映射

除了属性映射与继承映射之外，还需要处理的一种映射是对象之间的关系映射。对象之间的关系有：关联、聚合以及组合。对象之间这三种关系的映射方式一样的。

1、关系的类型

处理映射时，对象之间的关系可以分为两类。第一类是基于多重性的，包含三种类型：

一对一关系(One-to-one relationships)。这是一种两端多重性(multiplicity)最大值都为 1 的关系。

一对多的关系(One-to-many relationships)。也被称作多对一关系(Many-to-one relationships)，这种关系产生于一端多重性最大为 1，而另一端大于 1 的场合。

多对多关系(Many-to-many relationships)。这是一种两端多重性(multiplicity)最大值均大于 1 的关系。

第二类是基于方向的，包含两种类型：单向关系和双向关系。

单向关系。单向关系是指一个对象知道与其关联的其他对象，但是其他对象不知道该对象。

双向关系。双向关系是指关联两端的对象都彼此知道对方。

2、实现对象间的关系

对象之间的关系是通过对象的引用及操作来实现的。当多重性是 1(比如 0..1 或者 1)的时候，这种关系通过一个引用、一个 Getter()和一个 Setter()方法来实现。

多重性(multiplicity)为“多”(比如 n, 0..*, 1..*)的关系是通过集合属性以及操纵该集合的操作来实现的。

3、实现关系数据库中的关系

关系数据库中，关系是通过使用外键(foreign key)来维护的。外键的值可能是另一个表(某个行)键值的一部分，也可能就等于那个表(某个行)的键值。一对一关系中需要其中一张表来定义外键。

要实现一对多关系，需要定义一个从“一表”指向“多表”的外键，并且外键设在“多表”的一端。

在关系数据库中，可以采用关联表来实现多对多关联。关联表包含了与其相关联的表的主键的组合。这个基本技巧的要点在于将多对多关系转化为两个一对

多关系，它们都与同一个关联表相关联。

因为外键是用来连接表的，所以关系型数据库中的所有关系都是双向的。因此外键可以设在任意一个表上来实现一对一的关系。

4.4 系统的实现

4.4.1 开发环境

硬件环境：

CPU ： Intel Pentium IV 2.79G

内存 ： 512 M

浏览器 ： IE 6.0

网络带宽 ： 10M

软件环境：

开发工具 ： Eclipse-3.1 + Tomcat-6.0.13 + hibernatetools-3.1.0.beta5 插件

服务器 ： Tomcat-6.0.13

持久层 ： Hibernate 3.2.1

数据库 ： mysql 5.0.27

JDK ： JDK 1.5

还有一些相关的配置软件，可以参考有关的参考资料。

4.4.2 表示层实现

表示层的作用是为用户提供录入接口以及向用户展示操作结果。这里我们用 JSP 页面来实现。总体可以分为两大功能界面：

- 物料录入界面
- 物料结构界面(BOM 树)

物料录入界面又可以分为原材料录入界面、零件录入界面、部件录入界面和成品录入界面。这些物料的录入有一定的顺序性，如原材料的录入必须要在成品录入之前。

物料结构界面可以分为零件结构界面、部件结构界面和成品结构界面。它是用来展示和修改成品和半成品的结构的。只要数据库中已经录入了需要的物料，就可以使用这些物料构造您想要的产品结构。

表示层只是底层数据结构的一个展示，通过 Form 来规整其数据表单，实现起来没有困难，限于篇幅，在此不再赘述。

4.4.3 数据库设计

按照 BOM 元模型的结构设计，系统必须存储两类数据库表：一类数据库表用于存储实体信息，另一类数据库表用于存储 BOM 的结构信息。首先讨论实体库中的表设计。实体库中包含类实体表和类关系表。类实体表与层次类相对应，类关系表与类之间的关系相对应。类实体表如下面 4-1 所示。

表 4-1 类实体表

字段名	字段类型	是否可以为空值	默认值	字段说明
id	varchar(30)	否	主键，自增	零部件 ID
name	varchar(50)	否	无	名称
isRoot	boolean	是	False	是否为最终产品
costTime	float	是	0	组装时间花销
totalTime	float	是	0	总制造时间

此时，我们按照 3.4.5.1 中所讨论的映射关系，即整个类层次映射成单个表。本表用于保存产品、零部件和原材料的名称及其他属性详细信息。考虑到本系统仅用来测试，不考虑全部的属性以及不同 BOM 的特征属性的区别。

类关系表如下面 4-2 所示。

表 4-2 类关系表

字段名	字段类型	是否可以为空值	默认值	字段说明
id	varchar(30)	否	主键，自增	关系 ID
name	varchar(20)	否	无	关系名称
relationType	int	是	0(表示继承关系)	类之间的关系

mappingType	int	是	0(表示一对一)	映射关系类型
fid	varchar(30)	是	Null	类一的 id
sid	varchar(30)	是	Null	类二的 id
numRelation	float	是	0	组成的数量关系

其中，relationType 字段描述了两个类之间关系，取值可以为 0，1 和 2，分别代表继承，关联和聚合关系。mappingType 字段描述了两个类之间的对应关系，取值可以为 0，1 和 2，分别代表一对一，一对多和多对多关系。fid 和 sid 表示了相互关系的两个类，numRelation 描述了两个类之间的数量关系。例如，如果 fid 和 sid 之间是一对多的关系，fid 由三个 sid 组成，那么 numRelation 的数值为 3。

BOM 模型库用于保存 BOM 的构造信息，它是由类关系表来组成的，本设计中只包含了一个 BOM 模型表。由于每个 BOM 所包含的类关系不同，在关系数据库中还很难找到一种比较合适的模型表的构造方案，我们暂时使用冗余存储的方法，设置 20 个关系字段，因为只需要知道最顶层的构造关系，就可以通过遍历来获取 BOM 的构造。此外，BOM 模型表中含有 id 和 name 字段。

注意：这个系统使用的是 MySql 数据库。有一点是要特别注意的，在处理中文字符的时候，应该在数据库的加上末尾的一句声明，使得中文能够正确的录入和输出数据库。这个声明是“ENGINE=MyISAM DEFAULT CHARSET=utf-8”。

4.4.4 Hibernate 持久化层

Hibernate 是一个面向 Java 环境的对象/关系数据库映射工具。Hibernate 不仅仅管理 Java 类到数据库表的映射，还提供数据查询和获取数据的方法。这样就可以使程序员把精力放在更重要的程序设计上，而不是建立 JDBC 连接和释放 JDBC 连接上。

Hibernat 本质上还是包装了 JDBC 来进行数据库操作的，由于 Hibernate 在调用 JBDC 上面是绞尽脑汁的优化 JDBC 调用，并且尽可能的使用最优化的，最高效的 JDBC 调用，所以性能相当令人满意。Hibernate 是轻量级的封装，避免过多复杂的功能，减轻程序员的负担。同时，它也是一个开源的代码，提供开放

的 API，用户可以自行扩充其功能。Hibernate 在大多数主流 J2EE 应用服务器的受管理环境中都可以良好运作，也可以作为独立应用程序运行。

Hibernate 的精髓是持久层实现模式。它完全是针对对象的持久化，即把一个普通的 Java 对象映射到关系数据库中。面向对象设计中的继承与多态机制在 Hibernate 里也得到了支持，在数据查询中，它支持动态 Query，并提供对十六种数据库语言的支持，它沿用传统数据库的事务模型，使程序员不必为新的事务模型大伤脑筋。另外，Hibernate 是本地调用，比 Entity Bean 有更高的性能，而且它改进的速度之快也是其他 ORM 产品无法企及的。

Hibernate 是通过一些 xml 文件来实现对类的持久化的。首先介绍一下配置文件 hibernate.cfg.xml。它的通常结构如下：

```
<!--
  --Hibernate配置文件，负责建立数据库的连接和指定映射文件
  -->
<hibernate-configuration>
    <session-factory name="java:/hibernate/HibernateFactory">
        <property name="show_sql">true</property>
        <property name="hibernate.connection.driver_class">
            org.gjt.mm.mysql.Driver
        </property>
        <property name="hibernate.connection.url">
            jdbc:mysql://localhost:3306/test
        </property>
        <property name="hibernate.connection.username">
          Root
        </property>
        <property name="hibernate.connection.password">
          Jobs
        </property>
                    <property name="hibernate.dialect">
            org.hibernate.dialect.MySQLDialect
        </property>

        <mapping resource="bom/hibernate/Bomclass.hbm.xml" />
    </session-factory>
</hibernate-configuration>
```

不难看出，该配置文件中的 property 标签指定了建立数据库连接所需要的驱

动程序、数据库名称和类别等。这样，我们就可以连接到指定的数据库，从而对数据库中的数据进行操作。Mapping 标签则指定了 hibernate 映射文件，下面介绍一下该映射文件的结构。

```
<!--
  --Hibernate映射文件，负责将指定的类映射到数据库表
  -->
<hibernate-mapping>
    <class name="bom.hibernate.Bomclass" table="bomclass">
        <id name="id" type="java.lang.String" length="32">
            <column name="ID" />
            <generator class="uuid.hex"></generator>
        </id>
        <property name="pid" type="java.lang.String" length="32">
            <column name="PID" not-null="true">
            </column>
        </property>
        ...
    </class>
</hibernate-mapping>
```

其中，class 标签指定了要持久化的类以及它所对应的数据库中的表，它通常是一个实体类，或者更具体说是一个 POJO，类中只含有属性和 getter()、setter()方法。id 指定了与数据库表的主键相对应的类的属性，还包括了该属性的类型以及数据库表中该字段的长度等信息。Generator 标签用于指定主键的生成方式，本文是采用了 Hebernate 中自带的 uuid.hex 生成一个 32 位的键值，也可以自己另外写一个类来生成。下面的 property 标签就是把类中的其他属性与数据库中的字段对应起来。

4.4.5 持久框架的缓存设计与实现

一个系统将包括许多数据，可以对这些数据进行分一下类。

维护性数据：这是指那些在系统中的基本维护性数据，比如仓库里的物料组、客户类型、付款方式货币种类等等，这些数据存在的共同点是:数据量小，这种数据一般在 50 条记录以内，都是一些类型，分类之类的数据；字段少，这种数据的字段都是很少的，主要是 ID、名称、状态等；使用频繁高，这种数据的使

用率非常频繁，因为在其它表中作为外键，经常会被使用；维护少，这种数据一般在系统启用前期进行维护，大部分情况下不维护。因此，具有这些特点的数据，可以放到内存中进行缓存起来，建立“内存镜像”。

操作型数据：这种数据的特点是数据量比较大，字段也比较多，比如物料、客户等信息，不适合用到内存中。

日志型数据：对于数据量更多的，比如订单、订单明细、操作记录等，这就更不合适了。

建立了“内存镜像”的数据要注意数据的同步问题，也就是要实时保持内存中镜像与实际的数据库表数据一致，那么在进行更新(包括新增，更新，删除)时要同时更新内存表数据。

对于非 ORM 的开发模式来说，使用 Sql 语句进行数据访问的，要实现这种实时同步是很难的，采用了 ORM 的开发模式后，在 ORM 上，数据访问不是直接使用 SQL 的，而是通过中间的持久层来实现的。因此，这个持久层为实现这种同步提供了必不可少的条件。

在 O/R Mapping 时，标识一个实体为：需要保存到内存镜像，则持久层在进行此对象的 Save()、Update()、Delete()时，自动进行“内存同步处理”即可，而且这是完全可实现的。

另一方面，系统通过“内存镜像”实现的效率提高是建立在内存的牺牲上的，如果有过多的维护型数据使用“内存镜像”，那么，整个内存消耗将非常大，所以，在考虑使用多少“内存镜像表”时，是要考虑服务器的承受能力的。

在 ClassMap 中定义实体类的 IsSaveToMemory 为 true 即实现了该实体类的“内存镜像”功能。

<class name=”ComponentEntity” table=”Component” database=”BOMEntity” IsSavetoMemery=”true”>

在持久层框架中有个静态类，它的 ArrayList 里存放各种内存镜像表的镜像 DataTable，在进行 Retrieve()时，先判断此 ArrarList 中是否存在此实体的镜像，如果有的话，则直接从此“镜像 DataTable”中 Retrieve()出来，还有 trieveCriteria(获取标准)时，会从此镜像 DataTable 中使用 Seleet 符合条件的，生成新的镜像 DataTable 返回出来。

实体在进行 save()、delete()和 updateCriteria(更新标准)及 DeleteCriteria(删除标准)时都是会进行镜像 DataTable 的同步更新。

4.5 BOM 的关键技术实现

4.5.1 BOM 的校验

4.5.1.1 数据合法性校验

企业在实际生产中涉及的数据众多，牵扯面较大，不一定能很好的对所有的数据进行详尽的整理和输入。比如，设计数据来源于设计部门；装配件的装配提前期来源于工艺部门；采购件的提前期、成本来源于采购部门；子件在组装过程中的废品率来源于制造部门等等。在 ERP 中，新建的 BOM 虽然已经过了提交、审核、发布一系列操作，但 BOM 的数据量较大，靠人工核对的话，工作量大且容易出错。因此有必要对 BOM 相关数据的合法性进行校验。

BOM 数据合法性校验比较简单，主要可归纳如下：

1) 一个物料只能是实件、设计虚件、装配虚件中的一种；

2) 子件数量不能为空，也不能为 0；

3) 子件废品率不能为空，可为 0；

4) 子件来源或为自产，或为外购，不能为空；

5) 产品和自产部件的装配提前期、外购部件和零件的采购提前期不能为空，也不能为 0；

6) 每个物料的标准成本不能为空，也不能为 0。对于标准成本为空的部件，可以利用前面给出的计算部件成本的算法给出默认的参考成本值。

4.5.1.2 结构合法性校验

1) 环检测

我们已经知道，BOM 的基本结构是一个有向无环图，也就是说，无论是设计 BOM 还是标准 BOM，其父子关系所形成的路径是不能存在环的。如果存在环，不仅不符合现实世界产品的结构关系，而且有可能造成 BOM 分解算法的死锁，使得 ERP 系统不能正常运行，因此我们需要对这种情况进行检测。检查一

个有向图是否存在环要比无向图复杂。对于无向图来说，若深度优先遍历过程中遇到回边(即指向已访问过的顶点的边)，则必定存在环；而对于有向图来说，却不一定。

检测 BOM 结构中是否存在环，实际上是检查 BOM 结构是否满足 BOM 的基本结构定义中的约束条件 2、3，即给定 BOM 的基本结构(M, E, f)，对于条件 a) $\exists m \in M$，$<m, m> \in E$ 和 b) $\exists m \in M$，$<m, m> \in E^+$，检测这两个条件是否满足。这两个条件满足和有向图中存在环是等价的。

检测算法如下所示：

```
bool CheckCyc (sMaterialNo, iVersion, dsBOMStructure)
    // sMaterialNo 是要检测的产品编码，iVersion 是该产品的版本；dsBOMStructure
    是 DataTable 类型，存储该产品在 tBOMStructure 表中的结构信息
{
  ArrayList visited;
  If (Search (sMaterialNo, iVersion, dsBOMStructure, visited) return true;
  return false;

}

bool Search (sMaterialNo, iVersion, dsBOMStructure, visited)
{
  DataTable dsChildren;
  visited.Add (sMaterialNo);    //记录访问过的父件
  dsChildren=dsBOMStructure.Select(fParentNo=sMaterialNo and fPVersion=iVersion);
  if (dsChildren.Rows.Count!=0)
{
    for (int i=0;   i<dsChildren.Rows.Count;   i++)
{
          if (visited.Contains(dsChildren.Row[i]["fChildNo"])) return false; //如果存在
        环，则返回错误
  if(!Search(dsChildren.Row[i]["fChildNo"],dsChildren.Row[i]["fCVersion"],
dsBOMStructure, visited)) return false;
}
}
visited.Remove (sMaterialNo);   //如果 sMaterialNo 的所有子孙件已检查完毕，则删除
return true;
}
```

2) 节点类型检测

在构造产品族结构树时，也就是新建设计 BOM 结构时，我们必须保证：

1、如果一个非根节点是设计虚件，则其所在路径上的所有祖先节点均为设计虚件或装配虚件，且至少存在一个祖先节点是设计虚件。

2、一个非叶子节点的所有子件不可能全是可选件，或者说一个非叶子节点至少有一个子件是必选件。

即给定 $dBOM_i$，检测条件 a) $\forall m \in F_i$，$m \neq r_i$，如果$<m_0, m> \in E_i^+$，则 $m_0 \in F_i \cup V_i$，并且$\exists m_1 \in F_i$，$<m_1, m> \in E_i^+$；和 b) $\forall m \in F_i \cup V_i \cup S_i$，$|\{m_0 \in F_i \cup V_i \cup S_i | <m, m_0> \in E_i\}| \neq 0$，$\exists m_1 \in F_i \cup V_i \cup S_i$，有$<m, m_1> \in E_i \wedge f_{Ci}(<m, m_1>)$=main 是否满足。

4.5.2 总数量计算

在新产品提交、更改产品提交以及 MRP 运算前的 BOM 合法性检查等功能模块中，都会对产品 BOM 的相关物料计算总数量。因此，需要将总数量计算的算法作为关键技术来研究。总数量的计算是在最低层次码已经计算完毕的基础上进行的，计算步骤如下：

(1)定位于第一条记录(根结点)，置其总数量为 1，变量 N=1 记录最低层次码。

(2)将最低层次码为 N 的相关记录的 BOM 机器号存到数组。

(3)如果数组不空，依次数组中的 BOM 机器号：

① 查找相应的父项总数量。

② 计算本项物料的总数量：本项物料的总数量=父项需要量×父项总数量。

③ 如果数组的所有数据计算完毕，则 N—N+1，转(2)；否则，读取数组的下一个数。

④ 如果数组为空，则总数量计算完成。

4.6 面向对象的动态 BOM 模型与现有模型的比较

为了说明本文中的动态 BOM 模型的特点，我们从以下几个方面和传统 BOM 模型及可配置 BOM 模型进行比较。这两种模型更具有代表性，当然还有其他许多模型，在此我们不再一一列举分析。

(1) 适用范围

传统 BOM 模型是为每一种产品创建一个 BOM，即使是两种产品只是在颜色上或者是某一零部件有差异，也需要构造不同的 BOM，适用范围非常窄；可配置 BOM 模型是为某一产品族设计一个 BOM，即具有相似产品结构的产品共用 BOM，这就极大地减少了 BOM 的数量，拓宽了该模型的适用范围；动态 BOM 模型可适用于一切产品，元模型的引入使得所有产品都可以通过继承或者组合来得到自己需要的 BOM。

(2) 灵活性

传统 BOM 一旦创立以后就不能被改变，它只针对于规划好的即将生产的产品，无法被重用于其他结构中；可配置 BOM 模型首先抽象出所有相似产品的通用模型，可选项用参数变元表示，通过指定参数来配置所需的产品 BOM，灵活性比较强；动态 BOM 模型可以说是对 BOM 结构进一步解耦，不仅使得产品 BOM 的特性可以灵活的更替，其子结构也可以随意的变化。它可以较好的适应不同企业对 BOM 自定义的需求，同时，在企业的业务模型改变的情况下也可以显示良好的可变性。

(3) 技术实现难易程度

在传统 BOM 模型中，每个 BOM 是相互独立的，不需要考虑根据客户的需求的 BOM 和现有 BOM 的差异，因而在构造时也非常简单。无论是在时间上，还是在构造的准确度上都有较好的表现，这也是为什么这种模型仍然被广泛应用于制造型企业主要原因。可配置 BOM 模型需要解决的关键技术有模块化设计和参数化设计。在模型快化设计中要进行功能的分析和功能模块的划分，虚拟件的粒度是该部分实现的难点。而在参数化设计中要考虑各参数之间的制约关系，这对于动态性的实现非常困难，从已有规则中提取后的逻辑推导准确性不高，现今还没有一种很有效的解决方法。在动态 BOM 模型中，由各种零部件装配而成的

产品在逻辑上是一个整体，BOM 的设计和维护工作是建立零部件或产品列层次上的，设计者或用户无需关心怎样查询和维护产品或零部件的底层信息，因而设计和维护比较直观方便。

第5章 总结和展望

5.1 总结

随着市场竞争的日益激烈，传统的 BOM 已经无法满足客户个性化需求下的产品规模生产。传统 BOM 的结构和产品的耦合度很高，每一种新产品的出现都会导致 BOM 的“重新”设计，这使得好的设计框架无法重用，开发的效率非常低；同时，这也造成了大量的数据冗余，给数据的维护和扩充带来了困难。如何找到一种灵活的框架，来简化 BOM 的重构，降低结构数据量，加快执行速度是本文的主要研究内容。

论文从研究传统的 BOM 模型出发，分析现有各种模型的优点和缺点，在面向对象的模型设计中得到启发。考虑到面向对象技术的发展和关系数据库的成熟，本文把两者进行有效的结合，设计出一种 BOM 元模型。这种模型将 BOM 结构与实体之间的层次关系分离开来，能够利用已有的模型对新的 BOM 进行构造；另一方面，实体和实体之间的关系也被有效的分离，进一步降低了耦合度。在早期的相关研究中，面向对象技术与关系数据库存储技术之间存在着“阻抗不匹配”的情况，对象向关系表的映射难于实现，成为了本研究的一个瓶颈。对象关系映射(ORM)技术的出现很好的解决了这一问题。在业务层和数据层之间加入了 O/R 代理层，封装了与数据库相关的操作。本文还设计了一个小的系统用以实现该对象模型。根据数据录入测试显示，该模型具有较好的性能。

总的来说，本文对于 BOM 的重构问题做到了较为圆满的解决，取得了以下成果：

1、通过元模型的设计，提高了 BOM 的灵活性，消除了数据冗余，降低了开发成本。

2、简化代码，减少出错率。通过建立 ORM 系统，能够大量减少程序开发代码，开发数据层就比较简单，大大减少了出错机会。

3、隔离数据源，可以很方便的转换数据库。利用 ORM 可以将业务层与数据存储隔开，开发人员不需要关心实际存储的方式，数据库转换时也无需修改程序。

5.2 展望

元模型的设计和 ORM 技术的引入在构造 BOM 模型中发挥了较好的作用，适应了客户个性化下的规模开发的需要。但是由于现有的技术还不能对面向对象进行全面的支持，面向对象数据的表现力还不够丰富。该面向对象模型应该从以下两个方面进行深入的研究和发展。

1、元模型中的模型库构造。本系统中模型库的数据库表设计还没有得到很好的解决。如何在关系数据库中存储 BOM 模型所包含的实体关系是下一步的工作重心。对象关系模型只是解决关系和对象数据库矛盾的权益之计，只有对象数据库的成熟才能带来面向对象技术的繁荣。

2、对象缓存策略与查询优化。在关系数据库与面向对象技术之间加入一个对象持久层，提高了系统的清晰度，开发人员可以集中精力于业务层，而不需要考虑底层的存储机制，方便了程序开发。但是同时，由于这个中间层的出现，系统的性能必然受到影响，因此，在对象的持久层机制中，对象缓存策略方面的设计将是一个很重要的方面。另外，还可以对持久层产生的 SQL 查询语句进行优化，从而加快数据库响应时间。

参考文献

[1]. 周玉清，刘伯莹，杨宝刚，王新玲. ERP 原理与应用. 北京: 机械工业出版社，2002:33-42.

[2]. 王庆国，蔡淑琴等. 面向客户定制的动态 BOM 模型及算法.计算机工程与应用. 2002:430-450.

[3]. 朱玲，薛贺. 对象关系数据库系统映射模型及应用. 计算机工程与科学. 2007，29(12):120-122.

[4]. 曹礼廉，李芳芸. 面向对象 BOM 建模与设计. 信息与控制. 1994，23(5):315-320.

[5]. 李向东，范玉青，梅中义，巩应奎. 动态 BOM 在企业数字化中的组织应用.制造业自动化. 2003，25(8):42-44.

[6]. Peter Y.Wu, Kai A.Olsen, Per Saetre. Visualizing the Construction of Generic Bills of Material. VISUAL 2002, 2002, 302-310.

[7]. VanVeen E.A. Modeling product structures by generic bills-of-material. Elsevier Science Publishers, Amsterdam. 1992.

[8]. 石为人，彭世强，康静. 基于产品 BOM 的赋权有向图模型的实现. 重庆大学学报. 2004，27(3):41-44.

[9]. 石为人，张星，马振红，林荫华. 关系型数据库 BOM 表的遍历算法的改进及实现. 2005，28(7):82-85.

[10]. 程控，革扬. MRPII/ERP 实施与管理. 北京: 清华大学出版社, 2003:224-235.

[11]. Wolfram WoB. A rule-driven generator for variant parts and variant bills of material. In Proceedings of Eighth International Workshop on Database and Expert Systems Applications, IEEE Society, 1997,556-561.

[12]. Alexander Redlein. Fulfilling customer's needs by the use of a variant configurator for dynamic product definition. IEEE, 1999: 735-742.

[13]. 钱炜，赵亮，潘双夏. 大规模定制下产品配置实现方法研究. 机床和液压. 2005(4):83-85.

[14]. 王庆国，蔡淑琴，陈宏峰，石双元. 面向客户定制的动态 BOM 模型及算法. 计算机工程与应用，2002, (24): 94-95.

[15]. BOM. Glossary of Terms. http://www.feldmanengineering.com/BoM_Glossary.htm.

[16]. Jiao Jianxin, Tseng Mitchell M. A Methodology of Developing Product Family Architecture for Mass Customization. Journal of Intelligent Manufacturing,1999,10(1):3-20.

[17]. Xuehong Du, Jianxin Jiao and Mitchel M.Tseng. Graph Grammar Based Product Family Modeling. Concurrent Engineering: Research and Application, 2002, 10(2): 113-128.

[18]. Ahmad Khalaila, Frank Eliassen. An Efficient Bill-Of-Materials Algorithm. http://www.cs.uit.no/forskning/rapporter/Report/9731.html.

[19]. 洪名松，刘成颖，王先逵，李长城. 基于动态建模思想的物料清单柔性化定义. 清华大学学报. 2005，45(5):638-641,646.

[20]. Guoli, Gong Daxin, Freddie Tsui. Analysis and implementation of the BOM of a tree-type structure in MRPII. Journal of Materials Processing Technology, 2003(139): 535-538.

[21]. JI GUO, LI GONG DA, XIN, FREDDIE TSUI. A Tree Structuralized Storage Model of BOM. Journal of Systems Science and Systems Engineering, 2002, 11(1): 55-60.

[22]. CAMARINHALM,CARDOSOT. Selection of partners for a virtual Enterprise. Infrastructure for Virtual Enterprises. Networking Industrial Enterprises[C]Kluwer Academic Publishers, 1999: 259-278.

[23]. MohuaXiong, Shu Beng Tora, Li Pheng Khoob, Chun-Hsien Chen. A web-enhanced dynamic BOM-based available-to-promise system[J], International Journal of Production Eeonomics, 2003(84): 1 33-147.

[24]. 陈新林，黄奇.一种 BOM 的算法实现. 现代计算机. 2002(143):30-32.

[25]. 邵健，吕震，柯映林. 动态 BOM 在批量定制企业 PDM 和 ERP 系统集成中的应用. 组合机床与自动化加工技术. 2005(10):95-97.

[26]. 徐汉川，徐晓飞等. 一种结合设计 BOM / BOP 的制造 BOM 生成方法.中国机械工程. 2005，16(8):701-704.

[27]. 朱林，王俊，陈永府. 产品设计 BOM 的自动生成方法及关键技术研究.计算机工程与设计.2005，26(7):1833-1836.

[28]. 于晓，仲良维. 基于产品的 BOM 自动生成方法.精密制造与自动化.2006(3):55-57.

[29]. 张献乐，林逢升. 客户化 BOM 的模型设计. 计算机系统应用. 2007(5):91-92.

[30]. 余锐林，吴顺祥. 一种改进的 BOM 展开及低层码生成算法. 计算机工程与应用. 2005(27):100-102.

[31]. 刘艳凯，于明，张斌. ERP 系统中 BOM 构造方法研究. 计算机集成制造系统，2003，9(4): 309-313.

[32]. 张小剑，陈伟，窦延平. 关于 BOM 的数据库设计及算法的研究. 计算机应用与软件，2004，(5): 33-34.

[33]. 苟凌怡，魏生民. 基于关系型数据库的产品动态 BOM 的数据库设计与优化. 组合机床与自动化加工技术，1999 (5):6-9.

[34]. M.Vegetti, G.P.Henning, H.P.Leone. An Object-Oriented Model For Complex Bills of Materials In Process Industries. Brazilian Journal of Chemical Engineering, 2002, 19(4): 491-497.

[35]. Olsen K.A., Saetre P., Thorstenson A. A procedure-oriented generic bill of materials. Computers & Industrial Engineering, 1997, 32(1): 29-45.

[36]. Yu Liu, Jiajian, Mai, Lei Li. A Flexible BOM Structure Based on AND/OR Tree with Its Product Configuration and Property Verification. Software Research Institute, ZhongShan University.

[37]. Scott W Ambler. The Application Developer's Guide to Object Orientation and the UML(2nd Edition) [M]. England: Cambridge University Press, 2001.

[38]. J.C. Hernandez Matias, H.Perez Garcia, J.Perez Garcia, A.Vizan Idoipe. Automatic generation of a bill of materials based on attribute patterns with variant specifications in a customer-oriented environment. Journal of Materials Processing Technology 199(2008)431-436.

[39]. E.Muller, P.Dadam, M.Feltes. Efficient Assembly of Product Structures in Worldwide Distributed C/S Environments. Datenbank-Spektrum,2004,10(9):38-45.

[40]. M.Vegetti, G.P.Henning, H.P.Leone. An Object-Oriented Model For Complex Bills of Materials In Process Industries. Brazilian Journal of Chemical Engineering,2002,19(4):491-497.

[41]. Do Yun Koo, Soon-Hung Han, Soo-Hong Lee. An object-oriented configuration design method for paper feeding mechanisms. Expert Systems With Application 14(1998):283-289.

[42]. Bozarth, Cecil C. and Handfield, Robert B. Introduction to Operations and Supply Chain Management, Pearson Education, Inc., Upper Saddle River, New Jersey, 2006:461.

致谢

在中山大学软件研究所两年的学习和实践，是我求学生涯中获益最多的一段经历。在这期间，有很多的人给予过我帮助和支持，在此我表示衷心的谢意。

首先，我要感谢我的家人。他们给予我一个温馨舒适的成长环境，教育我成为一个正直的人，告诫我在顺境中要戒骄戒躁，鼓励我遇逆境时要不屈不挠，引导我在迷途时重回正轨。正是有他们无私的支持与付出，我才能全身心地投入到学习和工作中。

我也要感谢我的导师李磊教授。在我的学习和生活中，李老师给予了悉心的关怀和指导；此外，李老师为我提供了参加科研与工程项目的宝贵机会，使我收获良多。李老师活跃的思维、创新的思想、严谨的治学、正直的为人、自信的精神给我留下非常深刻的印象，这一切必将成为我一生中最宝贵的精神财富。

此外，我还要感谢 BOM 小组的各位师兄，他们是陈冰川、胡之斐和孙弘。他们与我在学习和科研上同甘共苦、共同进退。在我的论文写作过程中，他们也给我提出了不少的建议，为我带来巨大的启发。

我还要感谢软件所的邓克像老师和邓苑芹秘书，在学习和生活的各个方面，他们都给予了我很多有益的建议。

要感谢的人还有很多，在此无法一一列出。再次向所有关心、支持和帮助过我的师长、同学、朋友和亲人表示最衷心的谢意！