本体构建 101

为什么构建本体?

- 构建本体的原因主要有：
  - 在人或软件之间分享对信息结构的共同理解
  - 实现领域知识的重复使用
  - 作出明确的领域假设
  - 将领域知识与操作性知识分离
  - 分析领域知识

什么是本体?

- 在本文中，本体指对某个领域内的概念的正式且详细的描述，其包括：
  - **classes**：描述领域中的概念
  - **slots**：描述每个概念的属性
  - **facets**：描述属性的限制
- 一个本体及其 classes 的实例集构成一个**知识库**
- 实际应用中，构建一个本体包括：
  - 定义本体中的概念
  - 将概念进行分层，确定超类与子类关系
  - 定义概念的属性以及对这些属性的值的限制
  - 为实例填充这些属性值

本体构建方法

- 下面介绍一种可行的构建本体的方法
- 首先，需要强调本体构建中的三条基础准则：
  1. 不存在对一个领域建模的绝对正确的方法
     - 总是存在可行的替代方法
     - 最佳解决方法取决于具体的应用场景和预期的扩展性
  2. 本体开发必然是一个迭代的过程
  3. 本体中的概念应该与对应领域中的对象（物理或逻辑上）和关系接近
     - 这些最可能作为名词（对象）或动词（关系）出现在描述领域的句子中

第一步 确定本体的领域和范围

- 我们首先应该定义本体的领域和范围，即回答下列的基本问题：
  - 本体覆盖的领域是什么?
  - 我们要用这个本体来干什么?
  - 本体中的信息应该为何种类型的问题提供答案?
  - 谁将来使用并维护本体?

- 一种确定本体具体范围的方法是，列出一系列本体应该能够回答的问题
    - 将这些问题作为之后对本体的测试
        - 本体是否包含足够的信息来回答这些问题?
        - 答案是否需要特定级别的细节或特定区域的表示?

第二步 考虑重用现有的本体

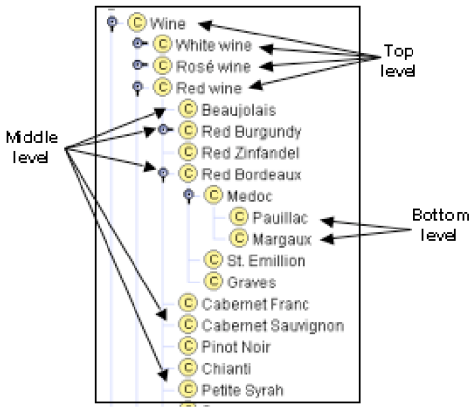
- 在从头开始构建本体之前，最好先调研是否有相关的本体已经被构建出来了
    - 我们可以基于这些本体进行进一步的改进和扩展
- 当前大部分本体都以电子形式提供，能够直接导入

第三步 列举本体中的重要术语

- 写下我们想要展示的所有术语的列表是很有用的
    - 一开始并不需要考虑术语之间的关系或概念重叠等问题，先保证列表的全面性
- 接下来的两步是开发类层次结构以及定义概念属性
    - 这两步是本体设计中最重要的步骤，也是联系紧密的两步
    - 我们通常先在层次结构中创建一些概念，然后描述这些概念的属性

第四步 定义类及其层次结构

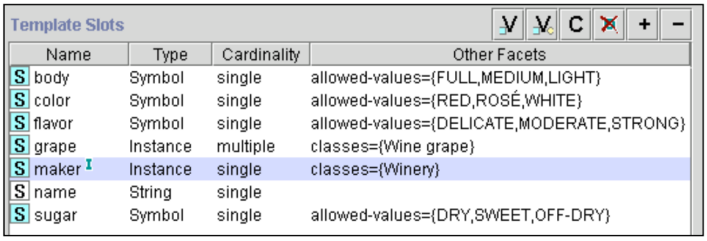
- 定义类及其层次结构通常有以下几种方法：
    - **自顶向下**的方法：先定义领域中最宽泛的概念，然后进行细化
    - **自底向上**的方法：先定义领域中最具体的概念，然后进行归纳
    - **混合**方法：将上述两种方法结合起来



- 上述三种方法没有优劣之分，取决于开发者对于领域的视角
- 不论使用哪种方法，我们都需要先定义类，再定义层次结构
    - 先从第三步中的列表中选取能描述对象独特性的术语作为类
    - 然后将这些类按层次结构组织
        - 如果一个类 A 是类 B 的父类，则每一个 B 的实例都是 A 的实例

第五步 定义类的属性

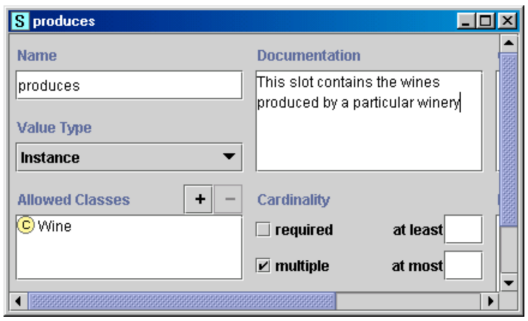
- 仅仅靠类无法提供足够的信息，我们还需要定义类的属性来进一步描述类



- 我们已经从第三步的列表中选择了术语来构建类
  - 大部分剩余的术语都是类的属性
- 一般来说，属性可以分为以下几种：
  - 内在属性：如酒的味道
  - 外在属性：如酒的名称、地域
  - 部件：针对结构化的对象，可以是物理或抽象的部件
  - 关系：与其他个体之间的关系（注意和类层次结构区分）
- 所有的子类都会**继承**其父类的属性
- 属性应该被放置到最宽泛的类中（即越靠近顶层越好）

第六步 定义属性的限制

- 类的属性有着许多限制（facets），如：
  - 如值的类型、允许的值、值的数量等
- 下面将介绍一些常见的 facets
  - **属性基数**：定义一个属性可以有多少个值
    - 有些本体只区分单数或复数的基数
    - 有些本体则进一步定义基数最大和最小值
  - **属性值类型**：描述一个属性的值的类型，常见的类型有：
    - String 类型
    - Number 类型
    - Boolean 类型
    - Enumerated 类型
    - Instance 类型
      - 用于表示个体之间的关系，需要定义允许的类的列表

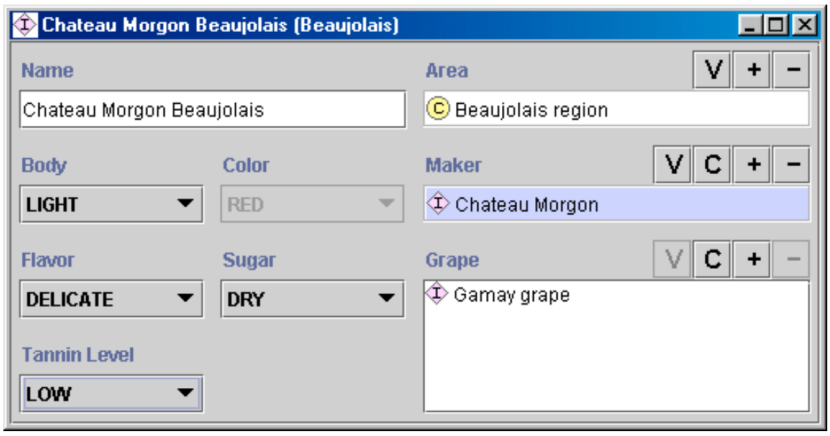


  - **属性的领域和范围**：
    - 对于 Instance 类型的属性，其允许的类的列表被称为属性的**范围**
    - 而一个属性所属的类被称为该属性的**领域**

        - 领域不需要单独指定，再设置属性时已经确认
    - 决定一个属性的领域和范围的法则是类似的
        - 尽量保持类的宽泛性，但不要过于宽泛

第七步 创造实例

- 最后一步是创造类的具体实例，步骤如下
    1. 选择一个类
    2. 创造该类的实例
    3. 填充属性值



定义类及其层次结构

- 本节介绍在定义类及其层次结构中可能会出现的错误
    - 虽然类的层次结构取决于具体的应用场景，但是存在一些通用的规则

确保类的层次等级是正确的

- 类 A 是类 B 的子类表示 A 为 B 的**一种**
    - 两者之间是 **is-a** 关系
- 一个常见错误是将单数的类作为其复数的子类
    - 两者间并不满足**"一种"**关系
    - 在命名时可以统一单复数来避免这一错误
- 等级关系具有**传递性**：
    - 如果 B 是 A 的子类，C 是 B 的子类，那么 C 是 A 的子类
    - 距离一个类最近的子类称为**直接子类**
- 类层次结构不是一成不变的
    - 可能随着时间的推移需要重构结构
- 要注意区分类及其名称
    - 类表示的是一类概念，其可能有多种名称
    - 相同概念的近义词不应该被表示成多个类
        - 很多系统允许为类关联近义词
        - 如果不能，应该在类的文档中说明
- 在构建类的层次结构时，还应该避免类的循环

    - 即 A 是 B 的子类，同时 B 又是 A 的子类
        - 这意味着 A 与 B 等价

分析类层次结构中的兄弟姐妹

- Siblings 指同一类的不同直接子类
    - 它们应该处于同一层级下
- 关于 siblings 的数量：一般在 2-12 个之间
    - 如果只有一个子类，说明建模可能有问题或本体不完整
    - 如果超过一打子类，则可能需要一个额外的中间类
        - 有时候，如果没有自然的中间类，那么可以保持原样（不需要人造类）

多重继承

- 一个类可以同时是多个类的子类
    - 它会继承所有父类的属性和限制

什么时候创建新的类

- 一般来说，如果子类包含超类所不具有的特征时，可以考虑创建，包括：
    - 有额外的属性
    - 属性有不同的限制
    - 参与了和超类不同的关系
- 在实际应用中，子类通常具有如下特征：
    - 新属性
    - 新的属性值
    - 新的属性限制
- 但是，有时即便子类没有上述特征，也应该创建：
    - 当类之间有术语层级结构关系时
        - 如疾病的分类，其可能没有属性或属性相同， 但也应该划分等级
    - 当领域专家普遍认为应该区分时
- 最后，我们不应该为每种额外的限制都构建子类
    - 需要在创建有用的新类和创建过多的类中间保持平衡

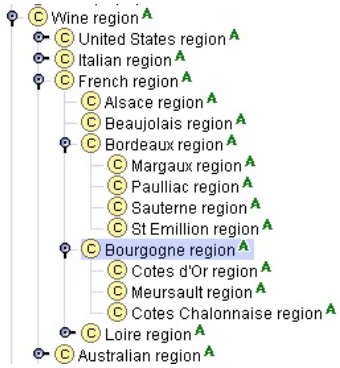
新的类还是新的属性值

- 有时候，我们需要考虑是创建新的类还是将其用不同的属性值区分
- 一般来说，应该遵循如下原则：
    - 如果不同属性值的概念会导致其他类中的属性值发生变化，那我们应该考虑创建新的类
        - 否则使用不同的属性值即可
    - 如果不同属性值的区别在你研究的领域中非常重要，并且你将不同属性值的对象看作不同种类的对象
        - 则应该考虑新建类
    - 如果一个独立实例所属的类会经常发生变化

        - 则那个类应该作为属性值
    - 一般数字、颜色、地点常作为属性值
        - 但酒是例外

实例还是类?

- 有时候，我们需要决定一个具体的概念是一个类还是单个实例
- 一般来说，应该遵循如下原则：
    - 单个实例是知识库需要表达的最具体的概念
        - 这取决与本体的应用目标
    - 如果概念形成了自然的层级，那么应该被表示为类
        - 因为没有子实例的概念
        - 即使表示为类后这些类不再具有直接实例
            - Protege 允许建立抽象类



限制范围

- 下面给出一些判断本体是否完整的规则：
    - 本体不应该包含领域中所有可能的信息
        - 不要过于具体，也不要过于宽泛（最多一层额外的等级）
    - 本体不应该包含所有可能的属性和类之间的关系
        - 只保留最关键的属性
        - 包含一些额外的关系信息会影响本体的使用
            - 如一个生物学试验本体，不能将试验者看做生物体的子类
            - 应该在文档中进行注明

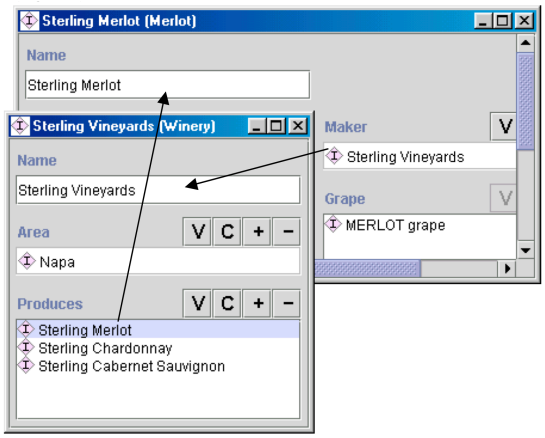
分离子类

- 许多系统要求在构建本体时明确哪些类是**分离**的
    - 即这些类不含有公用的实例
- 明确分离的类有助于验证本体的合法性

定义属性—更多细节

转置属性

- 一个属性的值可能会影响另外一个属性的值
- 例如制造者与制造的东西，两者之间相互关联
  - 已知 A 是 B 的制造者，则可以推断出 A 制造了 B
    - 这种关系称为转置关系
  - 虽然这种关系看上去有些冗余，但从知识获取的角度保留两者更加方便
  - 定义了转置关系后，知识系统会根据一个值自动填充另一个



默认值

- 大部分系统允许为属性指定默认值
  - 如果一个类的大部分实例都具有相同的某个属性值，则我们可以将其设置为默认值
- 默认值可以在该属性的限制内改为任意值
- 注意：要区分默认属性值与属性值
  - 属性值一旦确定是无法更改的
  - 如所有甜酒的糖度属性都为 SWEET，这并不是默认属性，不可修改

命名

- 在本体中，为命名制定规范有助于本体更加容易被理解，也可以避免一些建模错误
- 在制定命名规范时，我们需要明确系统的特征，例如：
  - 对于类、属性和实例是否共享命名空间
  - 是否大小写敏感
  - 名字中允许哪些分隔符

大小写与分隔符

- 对于一个本体中的命名，保持同样的大小写习惯可以提升其可读性
  - 通常将类名首字母大写，属性名首字母小写
    - 如果系统大小写敏感
- 当一个概念名称包含超过一个单词，我们需要进行分割，下面给出一些可能的选择：
  - 使用空格
  - 将单词连在一起，首字母均大写
  - 使用下划线、破折号或其他分隔符

        - 要决定每一个单词要不要首字母大写
- 一般如果系统允许，建议使用空格
    - 如果要和其他系统交互，还要考虑那些系统是否支持空格

单数或复数

- 在命名中使用单复数没有好坏之分（通常用单数）
    - 只要保持前后一致即可
    - 有的系统可能会提前要求确定单复数并不允许更改

前缀和后缀

- 有些系统会使用前缀或后缀来区分类和属性
    - 如 has- 或 -of
- 这种方法可以快速区分出类和属性
    - 只是名称会稍长

其他命名规范

- 不要在名称后再添加 class、property、slot 这样的字符串
    - 其他约束已经可以区分类和属性
- 尽量避免在名称中使用缩略词
- 对于一个类的所有直接子类，其名称中关于是否包括超类的名称应该统一

结语

- 本文介绍了如何构建一个本体
- 最重要的信息是：**任何领域都没有单一的正确本体**
    - 构建本体是一个创造性的过程
    - 本体的质量只能通过实际应用来评价