System Analysis and Design

Homework Assignment 4

学号：19335286

姓名：郑有为

1. List Some Key Ideas and Best Practices which Will Manifest in Elaboration.

1. 实行短时间定量、风险驱动的迭代
2. 尽早开始编程
3. 对架构的核心和风险部分进行设计、实现和测试
4. 尽早开始测试
5. 根据测试、用户和开发者的反馈进行修改调整
6. 通过每次迭代举行一次的讨论，详细编写大部分用例和需求

2. What is a Domain Model? Why Call a Domain Model a "Visual Dictionary"? How to Create a Domain Model?

- 什么是领域模型?

  领域模型是对领域内的概念类或现实世界中的对象的可视化表示[MO95, Fowler96]。

- 为什么成领域模型为可视化字典

  领域模型对领域的词汇和概念进行可视化和关联，领域模型所描述的信息也可以采用纯文本（UP词汇表）表示，而可视化语言中的概念和关系更易于理解，因此领域模型是可视化字典，表示领域的重要抽象、领域词汇和领域的内容信息。

- 如何创建领域模型

  以当前迭代中所有涉及的需求为界：

  1. 寻找概念类；
  2. 将其绘制为UML类图中的类；
  3. 添加关联和属性。

3. What are Conceptual Classes? How do I find Conceptual Classes?

- 概念类（Conceptual Classes）：通俗地说，概念类是思想、事物或者对象。更正式地讲，概念类可以从其符号、内涵和外延来考虑：
  - 符号：表示概念类的词语或图形。
  - 内涵：概念类的定义。
  - 外延：概念类所适用的一组示例。
- 例：考虑购买交易事件的概念类，可以用符号 Sale 对其命名。则 Sale 的内涵可以表述为：“表示购买交易的事件，并且具有日期和时间”。 Sale 的外延可以是所有销售的例子，即世界上所有销售实例的集合。
- 找到概念类的三条策略：
  - 重用和修改现有的模型。这是最首要、最佳且最简单的方法，若条件许可，通常从这一步开始寻找概念类。在许多常见领域中都存在已发布的、绘制精细的领域模型和数据模型，可以参考这些领域中出现的模型进行重用和修改，得到所需要的概念类。
  - 使用分类列表。可以通过制作概念类候选列表来开始创建领域模型。

- 确定名词短语。通过识别名词短语寻找概念类，即在对领域的文本性描述中识别名词和名词短语，将其作为候选的概念类或属性。但使用该策略要注意的是，不可能存在名词到类的映射机制，且自然语言中的词语具有二义性。

4. What are Associations in a domain model? How to find associations for a domain model?

- 什么是领域模型中的关联?

  关联是类之间的关系，表示有意义和值得关注的连接。在UML中，关联被定义为"两个或多个类元之间的语义联系，涉及这些类元实例之间的连接"。

  关联表示了需要持续一段时间的关系，根据语境，可能是几毫秒或数年。由于领域模型是从概念角度出发的，所以是否需要记录关联，要基于现实世界的需要，而不是基于软件的需要。

  关联被表示为类之间的连线，并冠以首字母大写的关联名称。关联的末端可以包括多重性表达式，用于指明类的实例之间的数量关系。关联本质上是双向的，无论从哪个类示例出发，到另一端的逻辑遍历都是可能的。

- 如何找到关联?

  通过列出需要考虑的所有类，然后根据实际情况去找类之间的关联。发生关联关系的两个类，类A成为类B的属性，而属性是一种更为紧密的耦合，更为长久的持有关系。可以从单向关联、双向关联、自身关联、多维关联等角度去寻找。

5. When to Define New Data Type Classes for attributes in domain Modeling?

当领域模型中的属性满足以下某些特征的时候，可以为其定义新的数据类型。

1. 由不同的小节组成，即一个属性是由多部分所构成，那么可以将这多部分看为一个整体，定义一个新的数据类型表示。如人名，通常是由姓和名组成；电话号码，通常是由国家代码、地区代码和用户号码组成。
2. 具有与之相关的操作，即在领域模型中存在着对某个属性的具体操作，如解析或检验等，那么可以为其定义一个新的数据类型，以便操作。如身份证号码、美国的社会安全号等。
3. 具有其它属性，即某个属性的内容除了应有的属性以外还可以包含其它属性以起到备注细化的作用，类似于C语言中定义结构体的思想，当我们需要用多个已有的数据类型来表达一个属性的时候，可以为其定义一个新的数据类型。此点与第一点有些类似，它们的区别在于，第一点所谓的"由不同小节组成"，通常各个小节有一定的次序要求，且各个小节通常是属于同一个数据类型；而第三点没有强调次序要求，更强调的是一种包含的关系，可以是由多种数据类型构成一个新的数据类型。具体例子如促销价格，除了价格这一个数值以外，还可以有促销的开始日期和结束日期以使其更加的具体。当然，也需要注意到属性的类型不应该是复杂的领域概念，属性不应该过于复杂，数据类型也自然不能过于复杂。
4. 单位的数量，如支付总额具有货币单位。
5. 具有以上性质的一个或多个类型的抽象。销售领域的商品标识符是诸如UPC和EAN这样的类型的泛化。

6. What is an System Sequence Diagram (SSD)? What is the Relationship Between SSDs and Use Cases? Why Use SSDs?

- SSD：即系统顺序图，是为阐述与所讨论系统相关的输入和输出事件而快速、简单地创建的制品，是对于用例的一个特定场景，外部参与者产生的事件，其顺序和系统之内的事件。它们是操作契约和（最重要的）对象设计的输入。
- SSD和用例之间的关系：SSD展示了用例中一个场景的系统事件，因此它是从对用例中的考察中产生的。

- 为什么使用SSD：在对软件应用将如何工作进行详细设计之前，最好将其行为作为“黑盒”来调查和定义。其中，系统行为描述的是系统做什么，而无需解释如何做，而这种描述的一部分就是系统顺序图SSD。

7. What Are Operation Contracts? How to Create and Write Contracts?

操作契约包含了以下部分：

- 操作：操作的名称和参数
- 交叉引用：会发生此操作的用例
- 前置条件：执行操作之前，对系统或领域模型对象状态的重要假设。这些假设比较重要，应该告诉读者。
- 后置条件：最重要的部分。完成操作后，领域模型对象的状态。

创建契约时可以应用以下指导：

- 从SSD中确定系统操作。
- 如果系统操作复杂，其结果可能不明显，或者在用例中不清楚，则可以为其构造契约。
- 使用以下几种类型来描述后置条件：
  - 创建和删除实例；
  - 修改属性；
  - 形成和清除关联。

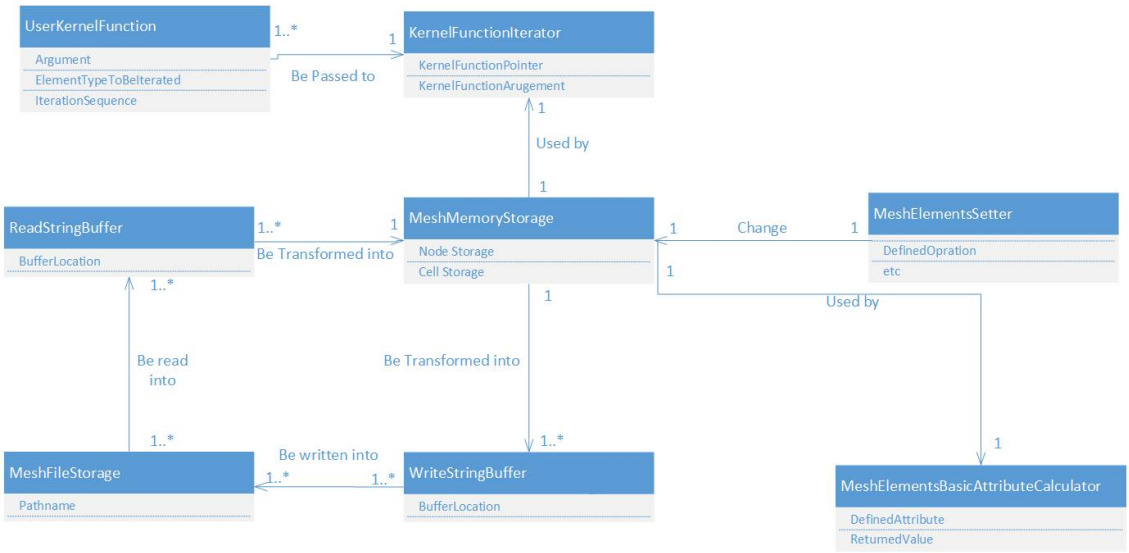
编写契约：

- 以说明性的、被动式的过去时态编写后置条件，以强调变化的观察结果，而非其如何实现的设计。
- 要在已有或者新创建的对象之间建立关联。

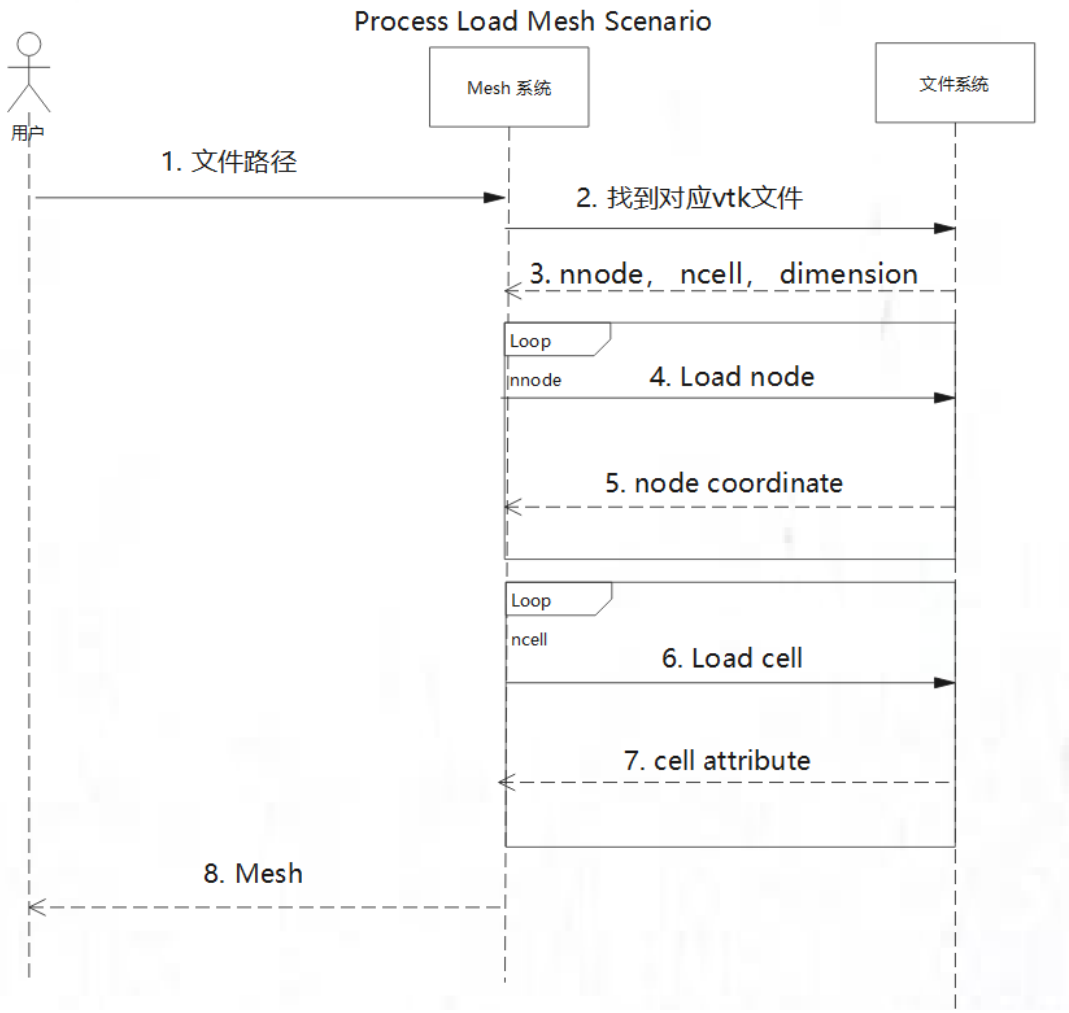
8. Revise the collection of all your artifacts you have done for your first iteration in your Elaboration phase of your project as in the following.

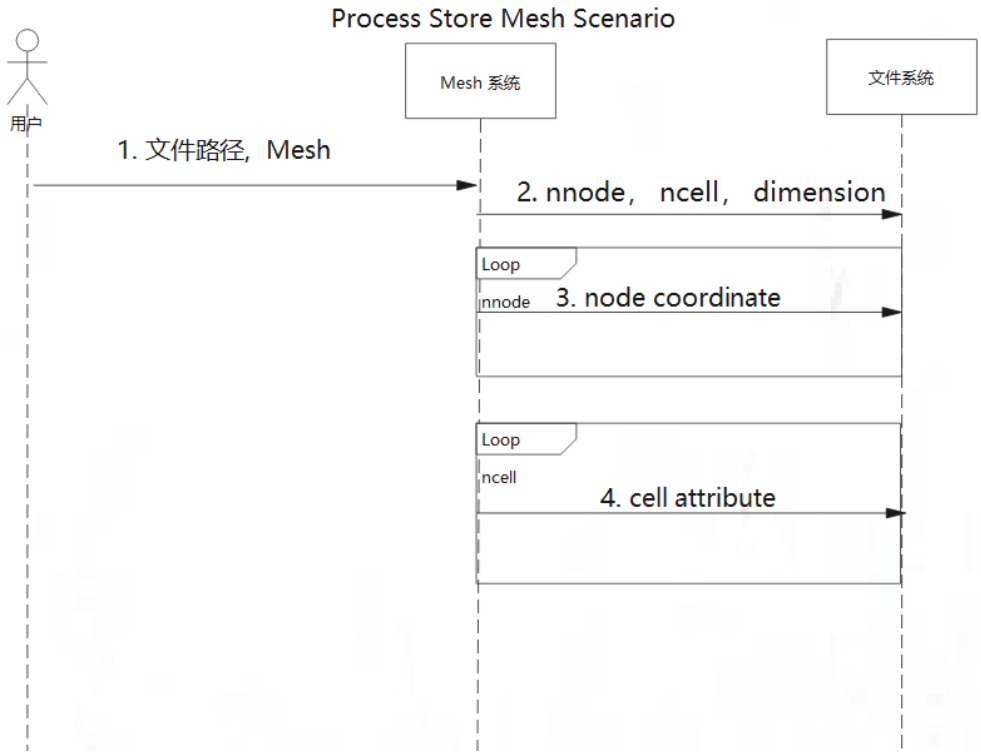
本次迭代更新以下内容：

- Domain Model (Conceptual Class Diagrams)



- SSD





- System Operation Contracts

  CO1: load_mesh_vtk

操作：load_mesh_vtk(const char* pathname)
交叉引用：用例：以给定格式读取网格数据文件并写入内存
前置条件：pathname所指向的空间已经被分配；文件存在且格式合法；
后置条件：Mesh对象中的node和cell向量被修改，原有数据被覆盖。

CO2：store_mesh_vtk

操作：store_mesh_vtk(const char* pathname);
交叉引用：用例：以给定格式将内存中网格数据写入文件
前置条件：pathname所指向的空间已经被分配；进程对pathname所指定的文件路径有写权限。
后置条件：在pathname所指定的文件路径中有新的（或者覆盖原有）vtk文件生成

CO3：parallel_looping

操作：void parallel_looping(void(*k_func)(void*, int), int* index, int index_size, void* arg, const char* element_name);
交叉引用：用例：迭代Kernel Function
前置条件：被迭代的核函数被定义；迭代的元素（element_name）被指定；
后置条件：取决于用户定义的核函数

CO4：serial_looping

操作：void serial_looping(void*(*k_func)(void*, int), int* index, int index_size, void* arg, const char* element_name);
交叉引用：用例：迭代Kernel Function
前置条件：被迭代的核函数被定义；迭代的元素（element_name）被指定；
后置条件：取决于用户定义的核函数