实验九 UML，逻辑， 软件体系结构设计（一）

郑家彤 202100032018

实验目的：

1. 深入理解UML

2. 了解计算机学科中的逻辑

3. 学习对比软件体系结构设计GB和IEEE最新SAD (Software Architecture Document)的标准

4. 研究经典软件体系结构案例

3. 完成自己项目的SRS

实验内容：

1. 阅读“The Unified Modeling Language Reference Manual”，进一步学习UML知识，理解如何应用UML对系统进行建模

理解UML的基本概念

静态结构：使用类图（Class Diagram）、对象图（Object Diagram）、包图（Package Diagram）、组件图（Component Diagram）、配置文件图（Deployment Diagram）等来描绘系统的静态组成部分，如类、对象、接口、组件及其关系。

动态行为：利用序列图（Sequence Diagram）、通信图（Communication Diagram）、时序图（Timing Diagram）、状态机图（State Machine Diagram）、活动图（Activity Diagram）等展示系统的动态交互、事件序列和状态变化。

环境与组织：通过部署图（Deployment Diagram）展示软件和硬件的物理布局，以及包图来组织和管理模型元素。

应用UML对系统进行建模

（1）需求捕获与分析

用例图（Use Case Diagram）：开始建模之前，首先识别并绘制用例图，明确系统外部参与者（Actors）与系统之间的交互，以及系统提供的功能（Use Cases）。

（2）设计阶段

迭代与增量：遵循迭代和增量的开发原则，逐步细化模型。从高层架构开始，逐步深入到详细设计。

类与对象：设计类图，定义类的属性、操作、关联、继承和实现关系，构建系统的静态结构。

交互与流程：利用序列图和活动图详细描述对象间的交互和控制流，确保需求的正确实现。

（3）包装与模块化

包图：通过组织模型元素到不同的包中，促进团队合作，管理复杂性，控制依赖关系，并支持版本管理。

（4）实现考量

组件与部署：组件图帮助设计系统的组件层次结构，而部署图则描述软件组件在硬件上的分布情况。

（5）验证与迭代

持续反馈：通过模型验证，检查设计的一致性和完整性。模型应该随着项目进展和新需求的出现而不断调整和优化。

2. 浏览“LOGIC IN COMPUTER SCIENCE--Modelling and Reasoning about Systems”，了解常用逻辑及其在计算机学科中的应用

（1）命题逻辑

基础概念：从声明性句子开始，介绍逻辑语句的基本构成和含义。

自然演绎系统：讲解如何使用自然演绎规则进行证明，包括基本规则和推导出的规则，以及证明等价性。

形式语言：将命题逻辑视为一个形式语言，定义其符号和结构。

语义：介绍命题逻辑的语义，解释逻辑连接词的含义，并通过数学归纳法展示其健全性和完备性。

范式：讨论逻辑表达式的正常形式，比如合取范式和析取范式，以及它们在证明中的作用。

（2）一阶逻辑

它是逻辑在计算机科学中应用的另一个关键部分，通常在命题逻辑之后被介绍。一阶逻辑允许变量、量化词（全称量词和存在量词）以及谓词，从而可以描述个体对象和它们之间的关系，这对于计算机科学中的数据结构、算法和数据库理论至关重要。

（3）谓词逻辑

谓词逻辑扩展了命题逻辑，允许使用变量和谓词来描述对象集合的性质，从而表达更复杂的陈述。

（4）谓词逻辑扩展

通过谓词逻辑，探讨变量、量词项和量化，以及如何在编程语言、数据库系统规范和类型理论中应用这些概念。

（5）时间逻辑

重点介绍线性时序逻辑(LTL)和分支时逻辑(CTL)，这些逻辑允许描述系统的动态行为随时间变化，广泛应用于硬件验证和通信协议的模型检查。

（6）模态逻辑

研究模态逻辑在人工智能和多智能体系统中的应用，处理知识、必要性、相信和可能性，对于理解和建模态之间的关系推理至关重要。

应用范围

软件验证：逻辑可以用来验证软件的正确性，通过形式化方法证明程序满足其规范。

知识表示：在人工智能领域，逻辑被用于构建知识库，表示实体、属性和关系，支持推理引擎进行自动推理。

数据库查询语言：SQL等查询语言的基础是关系逻辑，它允许用户以逻辑形式提出数据检索请求。

类型系统与编程语言设计：逻辑原理支撑着类型系统的设计，影响编程语言的表达能力和安全性。

模型检测与定理证明：自动化工具利用逻辑来检查系统模型是否满足特定性质，或者证明数学定理。

3. 分工协作，参考国标“13 - 软件(结构)设计说明(SDD)”等资料，对比参考SAD最新标准IEEE-42010.pdf，针对自己的项目设计SAD初稿。

（见小组SAD初稿）

4. 分工协作，学习、检索研究经典软件体系结构案例。

On-the-Criteria-To-Be-Used-in-Decomposing-Systems-into-Modules.pdf

http://www.cs.cmu.edu/~ModProb/index.html

软件体系结构风格：

1）管道和过滤器：构件被称为过滤器，负责对数据进行加工；连接件被称为管道，是输入数据流和输出数据流之间的通路。每个过滤器是一个独立的个体元素，各个过滤器互不相关；每个过滤器独立完成自己的任务，不同过滤器之间不需要交互。数据输出的最终结果与各个过滤器的执行顺序无关。

2）客户-服务器：系统被分为两部分，客户端负责用户界面和发起请求，服务器负责处理请求、执行业务逻辑和存储数据。

3）对等网络：每一个构件都只执行它自己的进程，，且对于其他同级构件，每个构件本身既是客户端又是服务器。每个构件都有一个接口，该接口不靠定了该构件所提供的服务，而且指定了它向其他同级构件所请求的服务。端与端之间通过彼此发送诸求的方式来实现通信。

4）发布-订阅：在发布-订阅的体系结构中，构件之间通过对事件的广播和反应实现交互。如果一个构件对某

事件感兴趣则可订阅该事件，一旦该事件发生了，另一个构件则进行发布来通知订阅者。发布-证所隐含的基础结构将负责注册订阅事件以及向合适的构件传达发布的内容。

5）信息库：信息库风格的体系结构由两类构建组成：中心数据存储以及与其相关的访问构件。共享数据存放于数据存储之中，而数据存取器是一个计算单元，它负责存储、检索以及更新信息。设计这样的系统是一个挑战，因为我们必须决定这两种类型的构件将如何进行交互。

6）分层：分层系统(layered system)将系统的软件单元按层次化组织，每一层为它的上层提供服务，同时又作为下层的客户。在一个“纯粹的”分层系统中，各层中的软件单元只能访问同层中的其他单元和相邻低层的接口所提供的服务。但是为了提高性能，在一些情况下这些条件也可能会放宽松，可以允许一个给定层访间所有低层的服务，这称为层次桥接(layerbrdging)。

7）KWIC索引系统：KWIC索引系统是软件体系结构研究领域的一个经典案例，它为我们提供了一个理解软件架构设计原则和实践的框架。在KWIC索引系统中可以采用使用共享数据的主程序/子程序、抽象数据类型/面向对象风格、基于事件通知的方案/响应式、基于管道的方案等解决方案。

5. 完成软件需求规格说明SRS

已上交

**下周五（含）前将软件需求规格说明提交给相应的助教**

项目跟踪，建立能反映项目及小组每个人工作的进度、里程碑、工作量的跟踪图或表，将其保存到每个小组选定的协作开发平台上，每周更新。