

# 需求文档结构化处理

# 模型与代码一致性报告

**团 队 名 称 TEAM**

**团 队 成 员 BY1817164 孙竖敬**

**团 队 成 员 SY1806116 赵健宏**

**团 队 成 员 SY1806611 张鹏程**

**团 队 成 员 ZY1806106 刘 良**

**2018年12月**

# 模型与代码一致性检查

## 一致性判别原则：

本次设计中的模型：用例图、类图、系统活动图、状态图、时序图、组件图、系统架构图、RUCM、OCL等。

本次设计中的代码：在flask架构下的工程

根据模型在指导代码构成中起到的作用，我们将对于以上所有模型进行检查。例如系统架构图，我们便会着重检查其是否与工程文件结构相同，并且代码中的调用是否符合该模型。

## 1.2 一致性检验：

### 系统架构图：

系统架构图中的各层均在工程的不同的文件夹中，名称均为该系统架构图的英文翻译，保证其与系统架构图的一致性。

* + 各层中的组件，
    - 响应层中的GWT导入器、领域背景导入、RUCM Generator分别为gwtresponse、backgroundresponse、rucmresponse模块。
    - 功能支持层的中间标签生成和自然语言处理分别为lablesupport和nlpsupport模块。
    - 数据支持层中的四个组件分别为datatype四个类。

### 组件图：

组件图已通过模型一致性的判别，所以我们可以将对于组件图的检查放在其他模型的检查中。（类行为的检查在类图中，组件的一致性检查在系统架构图中，对外接口的检查通过类图中的public进行检查，调用行为通过类图的调用进行检查。）

### 类图：

1. 对于类图中的大部分类，我们的系统工程中均有相对应的类，例如RUCM、TaggedGWT、NLPExecutor等。对于其他系统工程中没有的类，我们作为python模块或单独文件（html、纯文本文件）进行处理。
2. 其中每个类的继承和组成关系均通过python中的class进行了的表达。
3. 代码中的每个调用关系均符合类图中的关联。
4. 类图中的每个关联也在代码中通过读取文件，显示页面等方式进行表现。
5. 对于类图中所表达的访问权限，由于python语言本身的限制，我们尽可能进行了表达。

### 用例图：

代码在用例的实现上，与用例图保持一致性。但是对于两个actor，本次项目由于时间限制并未保证一致性。这是因为当前并未对于两个actor的权限进行限制产生的，只需增加人员类，区分两种身份并且对不同的身份提供不同的权限。

### 系统活动图：

该模型将系统活动通过更加直观的方式展现出来。对于活动图中每一个活动所在的甬道，都有与其对应的类或模块。对于每个活动之间的状态转移，其与代码是有相对应的代码块的。所以系统活动图与代码存在一致性。

### 状态图：

系统状态图与时序图有较深的联系，但是又稍有不同，其表示出系统不同状态之间的转化关系。但是在该系统中，若系统符合时序图，那么必然符合状态图。因为时序图中先导入领域背景再导入GWT然后生成TaggedGWT，保证了状态图中有领域背景和GWT的时候才能生成Tagged。而时序图中最后生成RUCM保证了状态图中领域背景与TaggedGWT与RUCM之间的同步关系。所以只需保证时序图与代码一致性就能保证状态图与时序图之间的一致性。

### 时序图：

系统时序图表示出系统进行工作时各个类的行为执行的前后关系。其中每个操作均有类图保证一致性。而我们的代码执行顺序与系统时序图保持了一致性，并且均按照同步操作进行，未出现并发操作。所以系统时序图与代码保持了一致性。

### RUCM：

RUCM表达了用户操作和系统行为。对于RUCM中每一行的活动，代码中均有对应的代码块，并且顺序与判断条件可以相对应，从而保证了一致性。

### OCL：

OCL与类图的一致性，让我们无需检查OCL每个元素与代码的一致性。我们现在只需检查OCL一些特殊的约束与OCL之间的一致性即可，即每个元素的约束和关联的约束。经过比对，我们的代码与OCL是具有一致性的。这不仅由后期检查保证，同时也由我们是通过OCL写代码，通过代码比对OCL检测是否实现功能保证。