

# 需求文档结构化处理软件设计报告

**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**

目录

[需求文档结构化处理软件设计报告 1](#_Toc533976182)

[一、系统概述 2](#_Toc533976183)

[二、架构设计 5](#_Toc533976184)

[1、系统架构设计 5](#_Toc533976185)

[2、类图主要类功能介绍 6](#_Toc533976186)

[3、组件图及相应组件层对应类图分析 8](#_Toc533976187)

[4、系统状态图 11](#_Toc533976188)

[5、系统时序图 12](#_Toc533976189)

[三、详细设计及软件展示 12](#_Toc533976190)

[1、OCL约束 12](#_Toc533976191)

[(1)展示层 12](#_Toc533976192)

[(2)控制层 13](#_Toc533976193)

[(3)响应层 14](#_Toc533976194)

[(4)功能支持层 18](#_Toc533976195)

[2、转换规则详细设计 22](#_Toc533976196)

[(1) 规约的GWT描述方式 22](#_Toc533976197)

[(2)标签设计规约 23](#_Toc533976198)

[(3) 转换规则说明 24](#_Toc533976199)

[3、软件样例展示 27](#_Toc533976200)

[3.1输入示例1： 27](#_Toc533976201)

[3.2 输入示例2： 30](#_Toc533976202)

[3.3 输入示例3： 33](#_Toc533976203)

[4、界面展示 34](#_Toc533976204)

# 一、系统概述

项目原始要求：给定自然语言文档，设计一套标签，手工嵌入到需求文档中，使其成为流程式的需求。并使用机器学习方法来自动插入标签。然后根据标签来提取信息，形成结构化的需求。

在此要求的基础上，明确给定的自然语言文档按照GWT形式进行描述，产生的结构化需求描述遵循RUCM格式并组成完整文档。

本项目立足软件需求描述方法，旨在建立一个将GWT形式描述的需求文档进行自动化处理转换为RUCM文档的实用系统。RUCM的需求描述方法相比GWT描述形式既保留了自然语言的表达能力与易懂性，又减少了表达的随意性与二义性。项目预期主要应用在软件工程的需求描述环节，在实现上要综合GWT需求文档与RUCM文档的特点设计文档标签，使用机器学习方法实现标签自动嵌入，由机器根据标签提取信息，转换成RUCM形式的需求文档。

项目要求的具体条目有

1. 给定自然语言需求文档：自然语言描述的需求文档是系统的输入数据。给定的自然语言文档以GWT形式描述。在软件开发过程中，需求文档记录了需求分析和导出的结果，是对用户和系统需求的全面详细的描述，自然语言的描述方法是最直观易懂且应用广泛的需求描述方法，但是自然语言的二义性使得这种方式有着明显的弊端，GWT格式的需求描述相比一般的自然语言描述方式准确性更高，但仍然可以转换成结构化程度更高的描述方式。
2. 设计一套标签：标签可以较高的层次上对需求文档的结构进行抽象，反映出软件需求的流程特性，在本项目中，需求文档标签是GWT描述与RUCM描述之间的桥梁，通过标签标注出GWT描述中与RUCM描述对应的实体与关系，在此基础上可以迅速地根据标签抽取信息产生RUCM文档。
3. 使用机器学习方法自动插入标签
4. 使用机器学习方法：使机器能够对自然语言描述的需求文档进行处理需要自然语言处理技术的支撑，在本项目中主要体现为自然语言的信息抽取（Information Extraction）过程，作为信息抽取的子过程，分词、标注、分块等文本处理操作都可以结合机器学习技术，通过引入机器学习方法可以得到准确性更高、灵活性更好的文档处理工具。
5. 自动插入标签：自动插入标签是项目中应用机器学习方法的目的，通过对自然语言描述的处理，在对描述内容达到一定理解程度的基础上可以由机器自行判断文档标签的位置，特别是在在处理规模较大的文档时可以拥有高效率。
6. 根据标签提取信息：标签完成了对原始的自然语言描述的结构划分，显示出了需求的流程特点。在为文档插入标签后，GWT描述中的RUCM内容被标记出来，根据标签可以快速提取出文档中对应于RUCM描述的内容。

形成结构化的需求：在通过标签提取出信息后，提取出的信息遵循RUCM描述格式，在此基础上，可以将信息保存为RUCM描述的需求文档，获得结构化的需求。

# 二、架构设计

## 1、系统架构设计

展示层

控制层

接收事件

响应层

GWT导入器

领域背景导入器

RUCM Generator

响应事件

获取支持

调用数据

数据支持层

领域背景

TagedGWT

GWT

调用数据

功能支持层

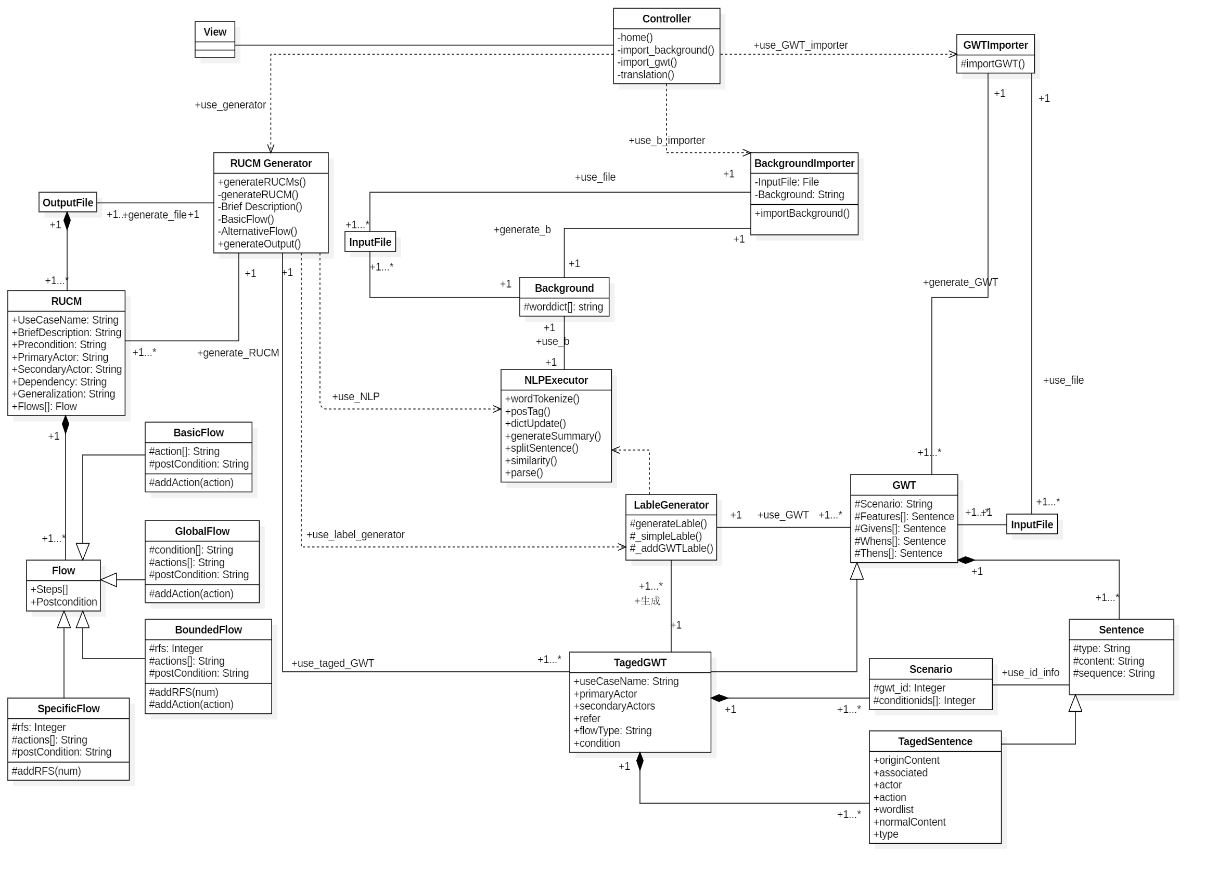
自然语言处理器

中间标签生成器

RUCM

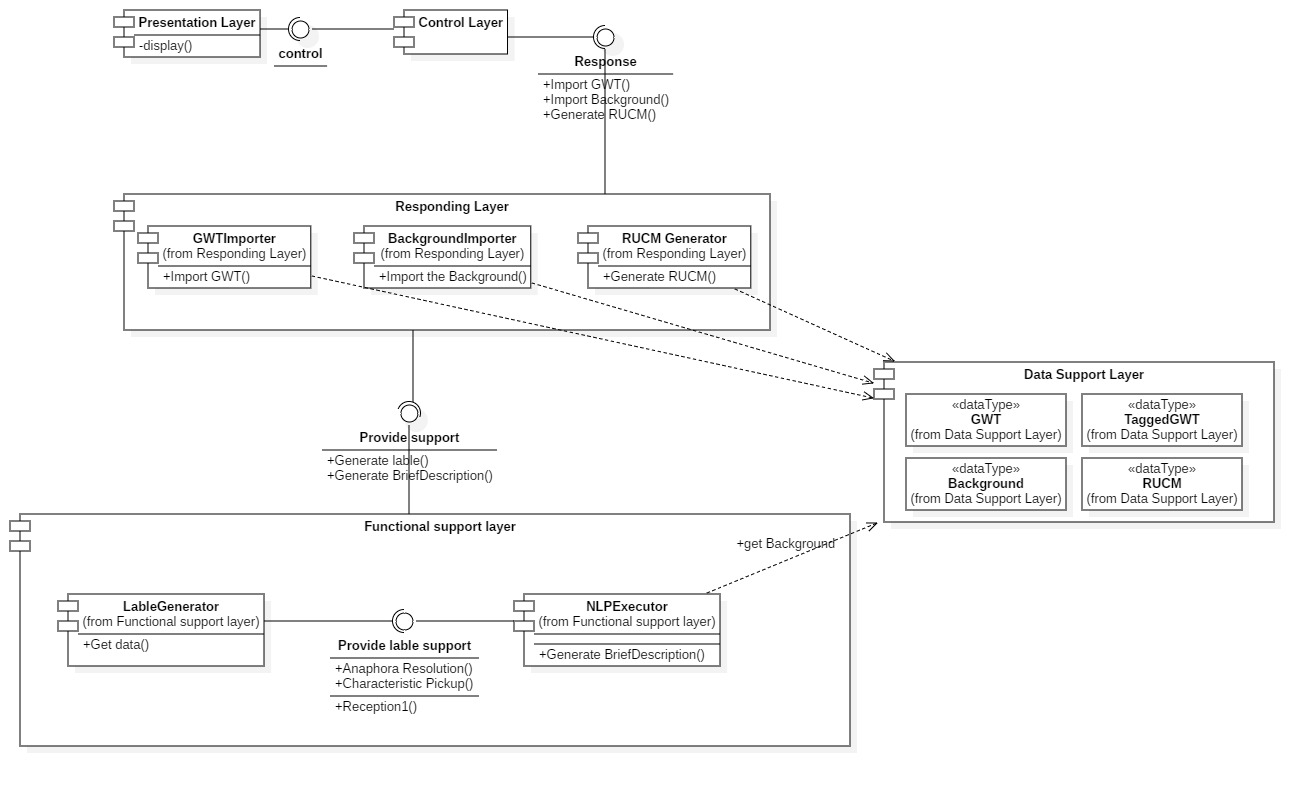
* + - 展示层：用于显示界面用于用户操作和查看
    - 控制层：接收展示层所产生的事件，并调用相应操作
    - 响应层：真正用于响应事件的实现
    - 功能支持层：提供支持用于响应层中组件实现响应
    - 数据支持层：存储数据

## 2、类图主要类功能介绍

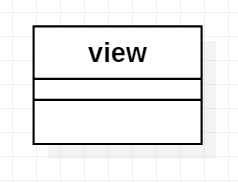


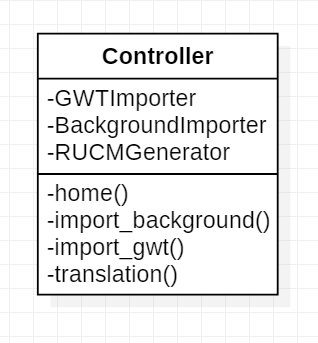
* + - Controller：负责相应外部事件，产生相应动作，负责调用RUCM Generator::generateRUCMs()，领域背景导入器::领域背景信息导入()，GWT导入器::导入GWT。
    - RUCM Generator：提供将GWT转化为RUCM的接口。
    - BackgroundImporter：提供导入领域背景的接口，将领域背景信息结构化存储。
    - GWTImporter：提供导入GWT的接口，将输入文档中的GWT文档，按照GWT类进行存储。
    - NLPExecutor：提供自然语言处理的一系列操作的接口，现在均为public为了使调用方便。但是在未来确认了自然语言处理器和RUCM Generator所使用的方法之后，可以进一步进行封装。
    - LableGenerator：将GWT转化为TagedGWT，便于接下来的完全转化过程。
    - Id：独一的表示出GWT中的语句。
    - Type：Feature、Given、When、Then的二级级标签。
    - Content：例：Given中的一行。或Feature中的一段话。
    - TagedGWT：添加标签后的GWT。在GWT的基础，添加GWT之间的关系，确定自身所在的Flow类型。每个GWT由Scenario进行表示，存储id和与当前GWT相关的condition的id。
    - TagedSentence：TagedGWT中的Sentence会变为TagedScence。该类除了GWT本身的内容意外，还存储二级标签下的三级标签以及句子之间的关系。
    - Association：句子之间的关系，首先需要由另一个句子的id，另外存储关系的类型，比如两句意思相同，或两句意思相反等。

## 3、组件图及相应组件层对应类图分析

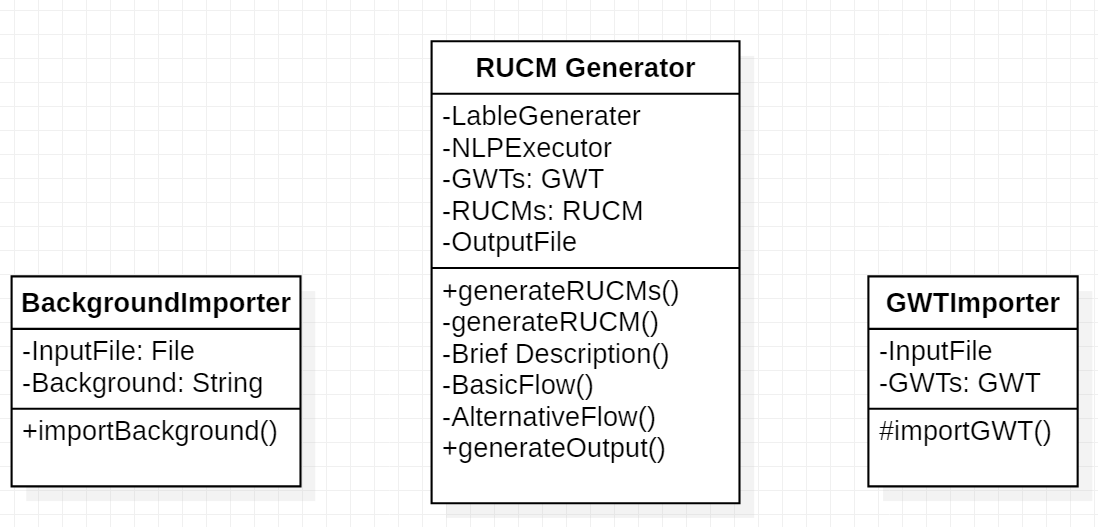


系统分为展示层、控制层、响应层、功能支持层、数据层，其中每个结构层对应着不同部分的类图，展示层中收到前端的输入，控制层收到相应信息，并调用相应的操作，响应层真正用于响应事件的实现，功能支持层对应着中间标签生成器和自然语言处理器用于提供支持用于响应层中组件实现响应；在最内部的数据支持层中对应着领域背景、TagedGWT、GWT。

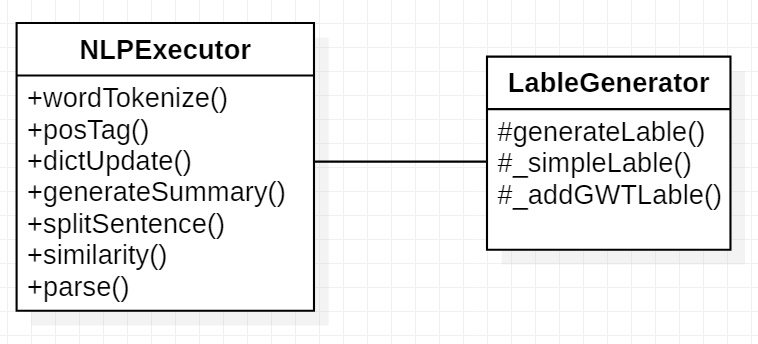




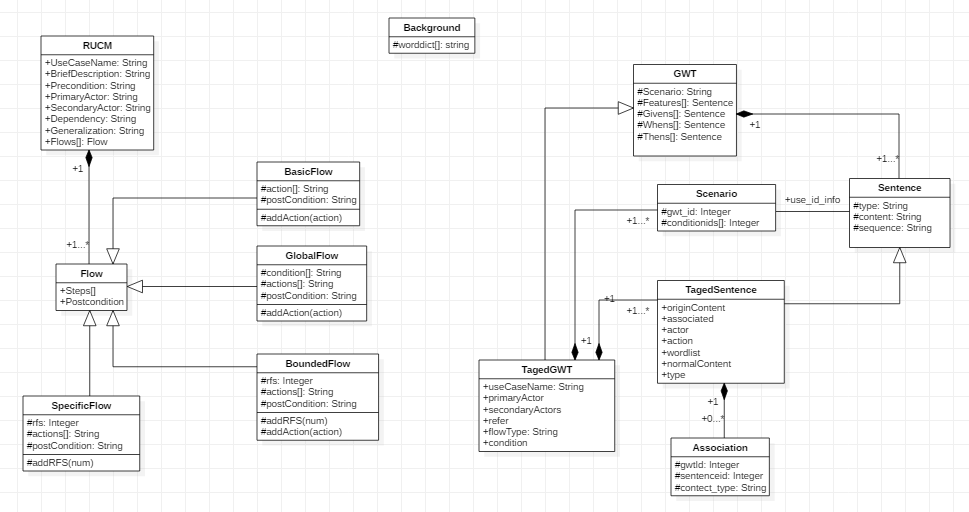
控制层控制GWT输入、领域背景输入以及RUCM生成的接口，对应着总体的三个用例。分别对应输入相应规则的GWT文档、GWT文档对应领域的数据库以及利用转换规则将GWT文档生成RUCM结构化文档。



该类图部分对应着组件图中的响应层，上面的控制层所控制的即是现在的响应层，响应层中的领域背景输入、GWT输入以及RUCM生成器都将分别利用数据层中的领域背景数据、GWT输入数据以及利用中间标签标记的GWT文档数据。

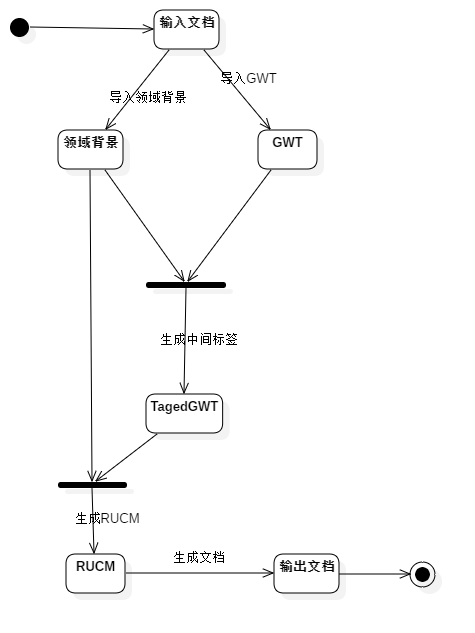


组件图中的功能支持层中对应着类图中的部分如上，该部分包含自然语言处理器和中间标签生成器。



数据层分为三类数据：领域背景数据、GWT描述数据、中间转换规则，用于提供相应功能的数据。

## 4、系统状态图



本系统的主要目的是文档的转化，而该状态图展示了在这个系统中文档的状态变化，声明了每种文档状态生成之前所需要的文档，表明了文档之间的相互依赖。

## 5、系统时序图



该图展示了整个系统的时序过程，展示了系统通过controller对整个系统的调度。同时中间标签生成器与自然语言处理器是通过递归方式进行调用，从而与controller隔离开来，减少了类与类之间的耦合关系，使得整个系统更加结构鲜明。

# 三、详细设计及软件展示

## 1、OCL约束

## (1)展示层

* 显示初始页面

context controller::home():file

pre:

post:return = index.html

### (2)控制层

* 调用领域背景导入，导入领域背景，显示导入结果

context controller::import\_background():(file, message)

pre: input\_file <> Null

index.html->领域背景导入->button->submit = true

post:if 导入成功

then return = ('index.html', message='领域背景导入成功')

else return = ('index.html', message='领域背景导入失败')

* 调用GWT导入，导入GWT，显示导入结果

context controller::import\_gwt():(file, message)

pre: input\_file <> Null

index.html->gwt导入->button->submit = true

post:if 导入成功

then return = ('index.html', message='GWT导入成功')

else return = ('index.html', message='GWT导入失败')

* 调用RUCMGenerator，首先选择需要导入的GWT，然后进行GWT->RUCM的转化，得到转化成功的RUCM，并且显示生成结果

context controller::translation():(file, message)

pre: index.html->开始转换->button->submit = true

post:if 选择GWT列表

then return = ('index.html', message=转化结果)

else return = ('index.html', message='已经取消转化')

RUCM.doc--或者这个不是在这里出现，而是在RUCMgenerator中出现

inv: notEmpty(转化结果)

### (3)响应层

* BackgroundImporter为领域背景信息导入器，将用户上传的领域背景信息传递给自然语言处理器，提高自然语言处理器的精度.

context: BackgroundImpoter

inv:self.NLPExecutor->size()=1

self.Background->size()>0

context: BackgroundImpoter::importBackground(segDict:file,posDict:file)

pre:segDict<>None or posDict<>None

self.NLPExecutor->size()=1

post: self.NLPExecutor<> self.NLPExecutor@pre

* GWTImporter为GWT描述导入器，将用户上传的GWT文件使用自然语言处理器NLPExecutor进行处理，提取主要内容以生成对应的GWT对象。GWTImporter每次读取一个用户输入的GWT文件。

context:GWTImporter

inv:self.inputFile->size()=1

self.NLPExecutor->size()=1

* 传入GWT文件路径或内存中保存的GWT文档内容，通过NLPExecutor提取出文档各字段内容转换成GWT对象。操作完成后GWTImporter关联的GWT数据对象数量增加。

context:GWTImporter::importGWT(filepath:string,filecontent:string )

pre:filepath.size>0 or filecontent.size()>0

self.NLPExecutor->size()=1

post: self.DataDispatcher.GWT -> size() >= (self.DataDispatcher.GWT -> size())@pre

* RUCMGenerator为RUCM描述生成器，对用户选择的GWT进行分析并添加中间标签，提取中间标签标记的内容生成RUCM对象，将一组RUCM对象输出为一个文本文档OutputFile。RUCMGenerator接受一组原始GWT对象，使用标签生成器LabelGenerator分析GWT对象属性并添加中间标签转换成TaggedGWT，使用自然语言处理器NLPExecutor分析一组TaggedGWT对象生成一个完整的RUCM描述。

Context RUCMGenerator

inv:

self.NLPExecutor->size()=1

self.LabelGenerator->size()=1

self.OutputFile->size()=1

self.RUCM->size()>=1

* 对输入的一组GWT数据进行分析，将其转换成一组对应的RUCM描述，生成输出文档内容。

Context RUCMGenerator::generateRUCMs(gwtList:set)

Pre: self.NLPExecutor->size()=1

self.LabelGenerator->size()=1

post: self.OutputFile->size()=1

self.RUCM->size()>=1

* 将输入的一组TaggedGWT对象转换成一个完整的RUCM描述。

Context RUCMGenerator::generateRUCM(taggedList:set)

Pre: self.NLPExecutor->size()=1

self.LabelGenerator->size()=1

taggedList->size()>0

post: RUCM -> size() = (RUCM -> size())@pre+1

* 由输入的一组TaggedGWT对象生成对应RUCM描述的Brief Description。

Context RUCMGenerator::briefDescription(taggedList:set，rucm:RUCM)

Pre: self.NLPExecutor->size()=1

taggedList->size>0

post:rucm.briefDescription<>None

* 由输入的一组TaggedGWT对象生成对应RUCM描述的Basic Flow。

Context RUCMGenerator::basicflow(taggedList:set，rucm:RUCM)

Pre: self.NLPExecutor->size()=1

taggedList->size>0

post: rucm.basicFlow<>None

* 由输入的一组TaggedGWT对象生成对应RUCM描述的Specific Alternative Flow、Bounded Alternative Flow、Global Alternative Flow(若RUCM含有这一种或几种可选事件流)。

Context RUCMGenerator::alternativeFlow(taggedList：set，rucm:RUCM)

Pre: self.NLPExecutor->size()=1

taggedList->size>0

post: rucm.specificAlt<>None or rucm.boundedAlt<>None or rucm.globalAlt<>None

* 将生成的一组RUCM对象转换成输出文件文本内容。

Context RUCMGenerator::generateOutput()

Pre: self.RUCM->size>0

post: self.OutputFile->size()=1

### (4)功能支持层

* LableGenerator为中间标签生成器，对输入的一组GWT进行分析并添加中间标签，转换成TaggedGWT对象。

Context LableGenerator

inv:self.NLPExecutor->size()=1

self.GWT->size()>0

* 对传入的一组GWT对象进行分析，添加标签生成对应的一组TaggedGWT对象

Context generateLable(gwtlist:set):set

pre:gwtList->notEmpty()

self.NLPExecutor->size()=1

post:self.TaggedGWT->size()>（self.TaggedGWT->size())@pre

* 对传入的一组GWT对象进行分析，添加GWT级的标签UseCaseName、FlowType，句子级的标签wordlist、actor、action

Context \_simpleLable(gwtList:set):set

pre: gwtList->notEmpty()

self.NLPExecutor->size()=1

post: self.TaggedGWT->forAll(useCaseName<>None or flowType<>None)

self.TaggedGWT.Whens->forAll(wordlist<>None and actor<>None and action<>None)

* 对传入的一组GWT对象进行分析，添加GWT级的标签refer、condition、FlowType，句子级的标签normalContent、type、associated

Context \_addGWTLable()

pre: gwtList->notEmpty()

self.NLPExecutor->size()=1

post: self.TaggedGWT->forAll(flowType<>None )

self.TaggedGWT.Whens->forAll(normalContemt<>None type<>None)

self.TaggedGWT.Thens->forAll(normalContemt<>None type<>None)

self.TaggedGWT.Givens->forAll(type<>None)

* NLPExecutor为自然语言器，向上为RUCMGenerator和LabelGenerator提供自然语言处理功能支持

Context NLPExecutor

inv:self.Background->size()=1

self.nlpTool->size()=1

* 对输入的中文句子进行分词，返回句子词汇的集合。

Context NLPExecutor::wordTokenize(sentence:string):set

pre: self.nlpTool->size()=1

sentence.size()>1

post:return=wordlist

* 对输入的中文句子分词结果进行词性标注，返回句子词汇词性标注的集合。

Context NLPExecutor::posTag(wordlist:set)

pre: self.nlpTool->size()=1

sentence.size()>1

post: return=wordtaglist

* 使用用户输入的自定义词典重新训练自然语言处理模型，可输入的自定义词典为分词词典和标注词典。

Context NLPExecutor::dictUpdate(segDict:file,posDict:file)

pre: self.Background->size()>0

self.nlpTool->size()=1

segDict<>None or posDict<>None

post: self.nlpTool<> [self.nlpTool@pre](mailto:self.nlpTool@pre)

* 将输入的两个句子向量化，计算并返回句子的相似度。

Context NLPExecutor::similarity(sent1:string,sent2:string):real

pre: self.nlpTool->size()=1

sent1.size()>0

sent2.size()>0

post:return=similarity

* 将输入的中文文本切分成句子，返回句子的集合。

Context NLPExecutor::splitSentence(text:string)：set

pre: self.nlpTool->size()=1

text.size()>0

post:return=sentlist

* 从输入的文本中提取重要信息，生成文本摘要。

Context NLPExecutor::generateSummary(text:string):string

pre: self.nlpTool->size()=1

text.size()>0

post:return=summary

* 根据句子的分词和标注结果做依存句法分析，返回句子的依存关系集合。

Context NLPExecutor::parse(wordlist:set , wordtaglist:set):set

pre: self.nlpTool->size()=1

wordlist.size()>0

wordtaglist.size()>0

post:return=parseset

## 2、转换规则详细设计

### (1) 规约的GWT描述方式

文件格式要求：utf8纯文本文件

使用者最低要求：掌握中文、掌握初等数学、理解分支与循环的含义、理解什么是用例、理解什么是用例的一个场景、对要做的用例有充分的相关知识。

GWT结构表述：

Scenario:\n <简单句>\n Feature:\n <肯定句>\n Given:\n (<简单句>\n)+ When:\n (<专用句>\n)+ Then: \n (<简单句>\n)+

GWT约束：

|  |  |
| --- | --- |
| 语言 | 语言说明 |
| \n | 换行，书写GWT时将\n的位置变为换行 |
| <?> | ? 为变量名称，即应该填写怎样的语句  书写GWT时，不需书写 |
| (?)+ | 表明 ? 出现一次或多次  书写GWT时，不需书写 |
| 简单句 | 书写GWT时，替换为具有以下性质的句子：  只含有一个主谓结构并且句子各成分都只由单词或短语构成的独立句子。  该句子在以下不同场景中，需要表达的含义不同：  Scenario后的简单句，应为对具体场景的简要描述。  Given后的每个简单句，应为该场景的前提条件。  Then后的每个简单句，应为该场景的结果，即为后置条件。  应存在一个且仅存在一个GWT，其前提条件应为整个用例的前提条件，其后置条件应为整个用例的后置条件。 |
| 肯定句 | 书写GWT时，替换为具有以下性质的句子：  陈述一个事实或者说话人的看法。  该句为肯定句。 |
| 专用句 | 书写GWT时，替换为具有以下性质的句子：  简单句，即只含有一个主谓结构并且句子各成分都只由单词或短语构成的独立句子，句子的主语为系统本身或用例的参与者即Actor。  以下每个……都为简单句，但是“如果”，和“直到”后面的状态可以用，“与”“或”分开。  使用“如果……，那么……，否则……”，表达完整的分支判断。  使用“如果……，那么……”，表达不存在不满足状态的分支判断。  使用“如果……，那么……，直到……”，表达循环。  在描述系统判断状态时，应使用“判断”，其前为系统的名称，其后为判断的状态。且该状态应与与该状态相反的状态采用同样的句式和相同的术语进行描述。  表述几个动作必须同时进行时，需要将多个简单句放在同一行以“同时”连接。  句子中若有具体场景的数据，应用于修饰宾语，作为宾语的定语。 |

### (2)标签设计规约

**UseCaseName**

**PrimaryActor**

**SecondaryActors**

**FlowType**

**Refer**

**TaggedGWT Condition originContent**

**Feature wordlist**

**Scenario actor**

**Given Sentence action**

**When normaContent**

**Then type**

**Associated**

对应每一个GWT对象生成一个标签化的TaggedGWT，在TaggedGWT级别有

* **UseCaseName**

GWT所描述的用例名称

* **PrimaryActor**

GWT描述场景的主要参与者

* **SecondaryActors**

GWT描述场景的次要参与者，是由参与者组成的集合

* **FlowType**

表明GWT描述场景的类型，与RUCM描述中的事件流类型对应，可能的标记为

**basic , specific , bounded , global**

其中标记为basic的GWT描述的场景是用例中各步正常执行得到理想结果，标记为specific的GWT描述的场景是用例中某一个步骤出现异常时所做的处理，标记为bounded的GWT描述的场景是用例中某几步中的一步或几步出现异常时所作的处理，标记为global的GWT描述的场景是用例中任何情况下出现满足GWT前提条件的事件时所做的处理。

* **Refer**

标记描述异常处理场景的GWT所对应的发生异常的步骤。

对FlowType标签标记为specific的GWT，其Refer记录了此GWT描述场景对应的步骤序号；对FlowType标签标记为bounded的GWT，其Refer记录了此GWT描述场景对应的步骤序号的集合。

* **Condition**

记录异常处理场景发生的前提条件。

对FlowType标签标记为specific的GWT，其Condition记录了此GWT描述场景发生的单一前提条件；对FlowType标签标记为bounded的GWT，其Condition记录了此GWT描述场景发生的可能的前提条件的集合；对FlowType标签标记为global的GWT，其Condition记录了此GWT描述场景发生的前提条件。

同时包含原始GWT中的信息

* **Feature**
* **Scenario**
* **Given**
* **When**
* **Then**

对Scenario，Given，When，Then中的句子都有如下标签

* **originContent**

句子的原始内容，是一个字符串。

* **wordlist**

句子的分词词链，是句子中词的集合。

* **actor**

句子的主语，是句子动作的执行者。

* **action**

句子的谓语，是句子的核心动作。

* **normaContent**

规范化的句子内容，对含有具体数据的句子将其规范化为符合RUCM要求的句子，对描述条件和循环的语句，使用RUCM的关键词进行替换。

* **type**

标记句子的类型。

对Given中的句子，有common , unique两种类型，common表示整个用例的前提条件，unique表示GWT所描述场景的特有前提条件。

对When中的句子，有normal ，conditional , circular三种类型，分别表示When中的通常语句、条件语句和循环语句。

* **associated**

标记与句子相关的操作索引，仅用于标记Given中Type为unique的句子，指向与该句子相关的具有basic标记的GWT的When中的操作。

### (3) 转换规则说明

1、从文本到GWT

从纯文本文件读入原始的GWT文本，从Feature关键字处切分文本，划分出每个GWT的文本， 使用正则表达式提取出GWT每个字段下的文本，将Feature文本记录到GWT对象的Feature属性，对其他字段文本，首先执行分句操作，获取每一句的原始文本，生成对应的Sentence对象，将每个Sentence对象保存到其归属的GWT属性中。

2、从GWT到TaggedGWT

对一组具有相同Feature值的GWT对象，每一个GWT对象创建一个对应的TaggedGWT对象，将GWT对象的属性值复制到TaggedGWT的对应属性中。

对每一个TaggedGWT对象，首先标记基础标签，包括GWT级的UseCaseName和FlowType的basic或bounded标签，句子级的wordlist、actor和action标签。分析所有TaggedGWT的Given属性中句子的数量，将其中句子最少的TaggedGWT的FlowType标记为basic，句子数量超过标记为basic的GWT 2句或以上（如果有）的TaggedGWT的FlowType标记为bounded。对一组中的每一个TaggedGWT，从TaggedGWT的Feature字段读出用例名记录到UseCaseName属性中，对When属性中的每一个Sentence对象，首先执行分词操作，将分词结果保存到句子的wordlist属性，使用句子的分词结果对句子做依存句法分析，根据依存句法分析的结果搜索句法树，将依存关系为主谓关系的词记录到Sentence对象的actor属性，将依存关系的核心词记录到Sentence对象的action属性。

完成基础标签标记后，进一步标记GWT级FlowType中的specific和global，refer标签，condition标签；句子级的normalContent，type和associated标签。

对TaggedGWT的When中记录的句子，根据其句法分析结果判断是否为简单句，若为简单句，将其type标记为normal，否则分析其wordlist中的关键词，若含有表示条件的关键词则标记type为conditional，若含有表示循环的关键词则标记type为circular，在此基础上将这些句子的文本规范化，对normal类型的句子去掉表示具体数据的宾语的定语，生成规范化句子内容normalContent，对类型为conditional和circular的句子将其关键词替换为RUCM关键字，生成规范化的句子内容normalContent。

对TaggedGWT的Then中记录的句子，分别进行分词和依存句法分析，填充标签wordlist，对句子的表述做规范化，填充标签normalContent.

对FlowType标记为basic的TaggeGWT，将其Given中句子的type标签标记为common，对FlowType标记为bounded的TaggedGWT，将其Given中的句子与basic TaggedGWT的Given中的句子做相似性比较，找到其中相近的句子标记type为common，其他句子的type标记为unique.将标记为unique的句子与basic TaggedGWT中When记录的句子相似性进行比较，对其中相似的句子，将unique句子的associated标记为该When中句子的索引。

对未标记FlowType的TaggedGWT，将其Given中的句子与basic TaggedGWT的Given属性记录的句子进行相似性比较，与type为common的句子相似的句子type标记为common，其他的句子type标记为unique，将unique的句子添加到TaggeGWT的condition属性中。在此基础上，找出basic TaggedGWT的When属性中与unique句子相似度最高的句子，若两个句子是相近的，则将此TaggedGWT的FlowType标记为specific、refer标记为相近句子的序号，并将句子的associated标签标记为预期相近的句子的序号；若两个句子不相近，则将此TaggedGWT的FlowType标记为global，将unique的句子记录到此TaggedGWT的condition属性。

3、从TaggedGWT到RUCM

根据UseCaseName相同的一组TaggedGWT生成一个RUCM对象，依次为

Use Case Name：使用TaggedGWT的UseCaseName

Brief Description：遍历一组TaggedGWT，将所有TaggedGWT的Scenario句子中的originContent组合成一个文本，使用TextRank算法对文本进行摘要生成，将生成的摘要文本作为RUCM的Brief Description.

Primary Actor , Secondary Actors:对所有TaggedGWT的When属性中句子的actor，统计所有actor的数量，去掉其中表示系统本身的actor，选择剩下的actor中数量最多的作为RUCM的Primary Actor，其余的actor组成RUCM的Secondary Actors.

Precondition: 使用FlowType标记为basic的TaggedGWT，将此TaggedGWT的Given属性中句子的originContent属性值连接成RUCM的Precondition.

Basic Flow：使用FlowType标记为basic的TaggedGWT，将此TaggedGWT的When属性中句子的normalContent依次作为Basic Flow中的action，将此TaggedGWT的Then属性中句子的normalContent连接成Basic Flow的postcondition。

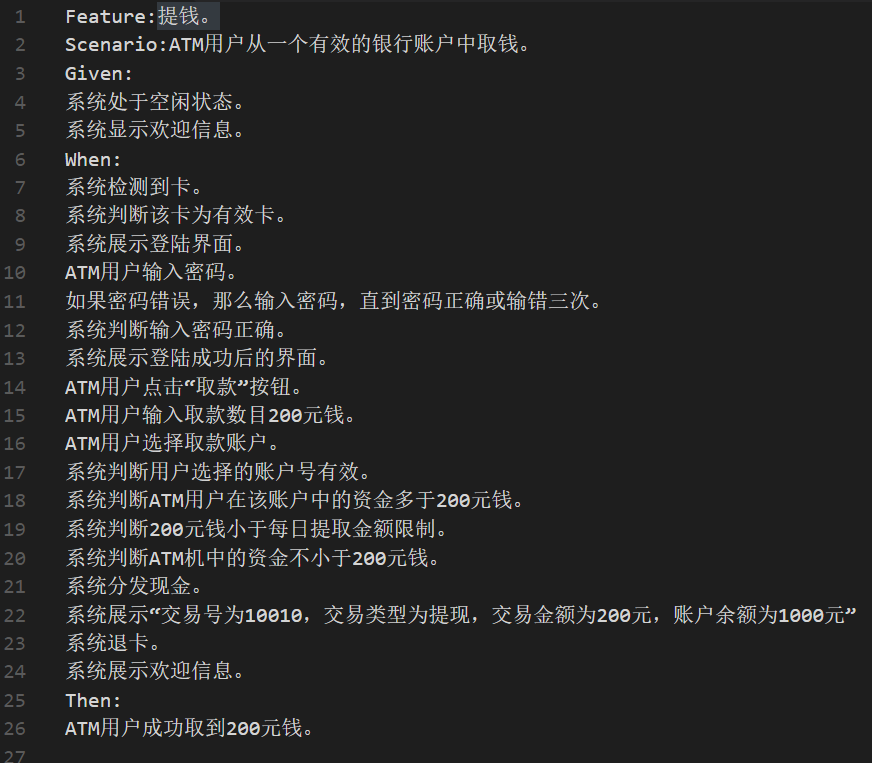
Specific Alternative Flow：使用FlowType标记为specific的TaggedGWT（如果有），将此TaggedGWT的Refer属性值作为Specific Alternative Flow的RFS，将此TaggedGWT的When属性中句子的normalContent依次作为Specific Alternative Flow中的action，将此TaggedGWT的Then属性中句子的normalContent连接成Specific Alternative Flow的postcondition。

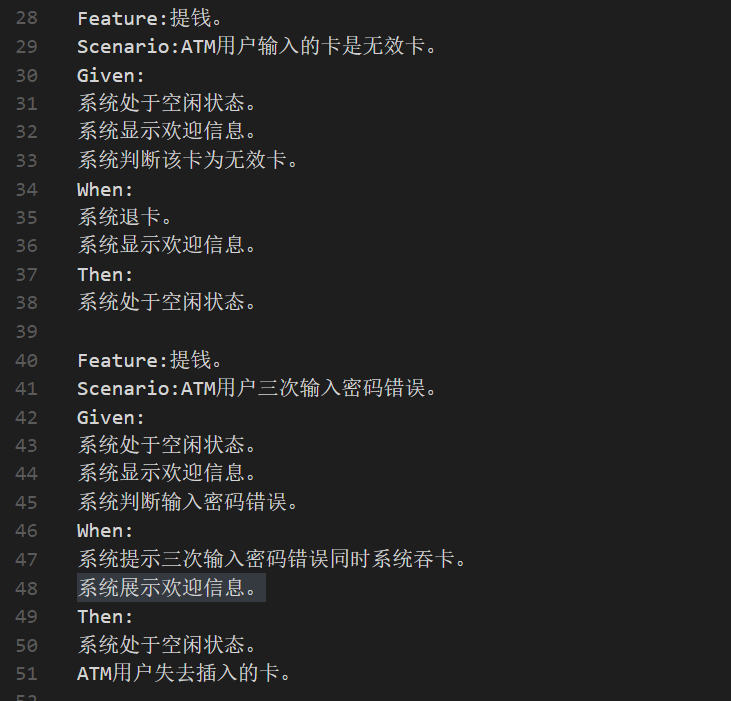
Bounded Alternative Flow：使用FlowType标记为bounded的TaggedGWT（如果有），将此TaggedGWT的Refer属性值作为Bounded Alternative Flow的RFS，将此TaggedGWT的When属性中句子的normalContent依次作为Bounded Alternative Flow中的action，将此TaggedGWT的Then属性中句子的normalContent连接成Bounded Alternative Flow的postcondition。

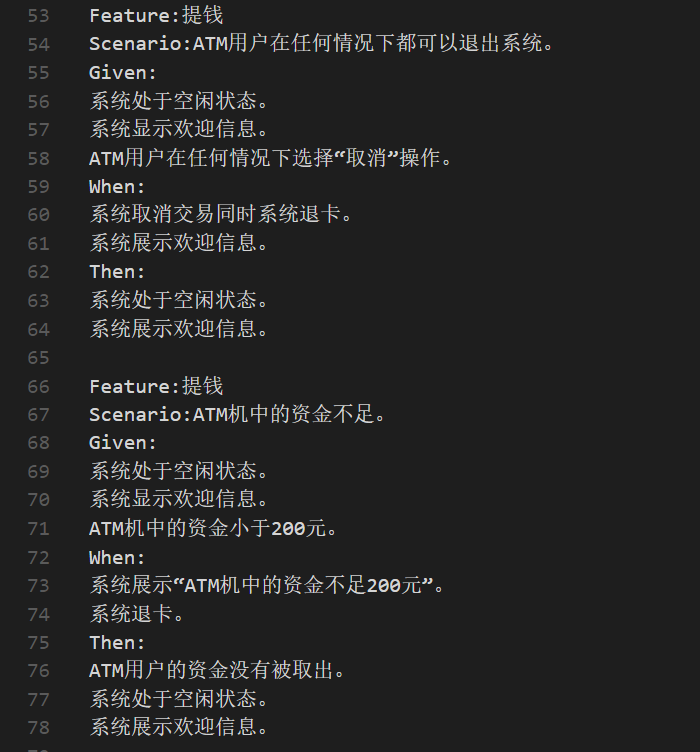
Global Alternative Flow：使用FlowType标记为global的TaggedGWT（如果有），将此TaggedGWT的condition属性值作为Global Alternative Flow的发生条件，将此TaggedGWT的When属性中句子的normalContent依次作为Global Alternative Flow中的action，将此TaggedGWT的Then属性中句子的normalContent连接成Global Alternative Flow的postcondition。

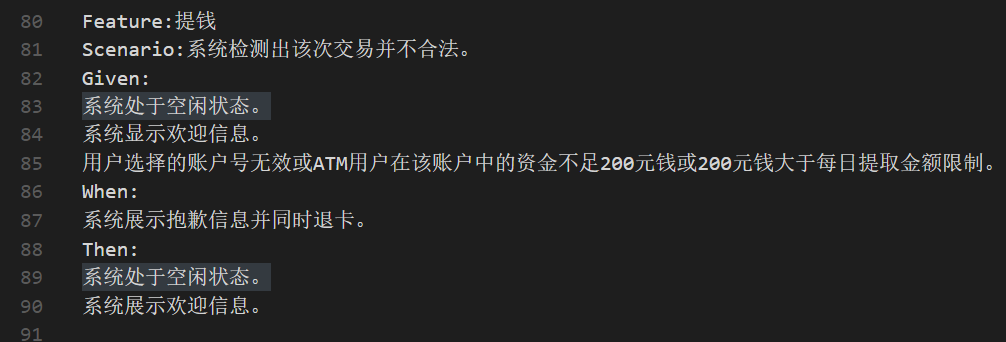
## 3、软件样例展示

## 3.1输入示例1：

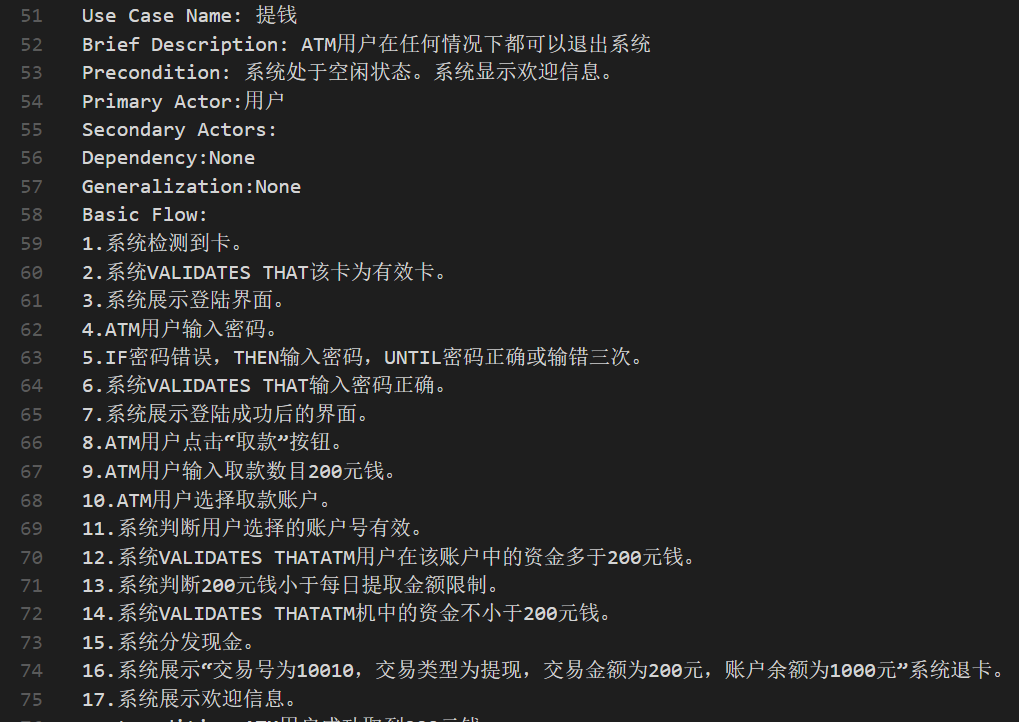


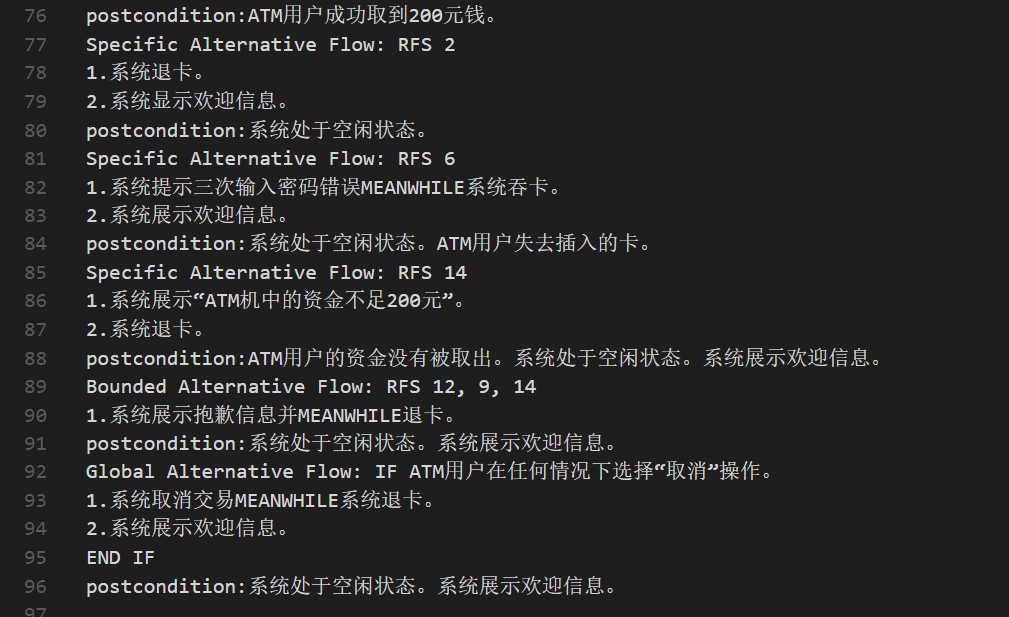




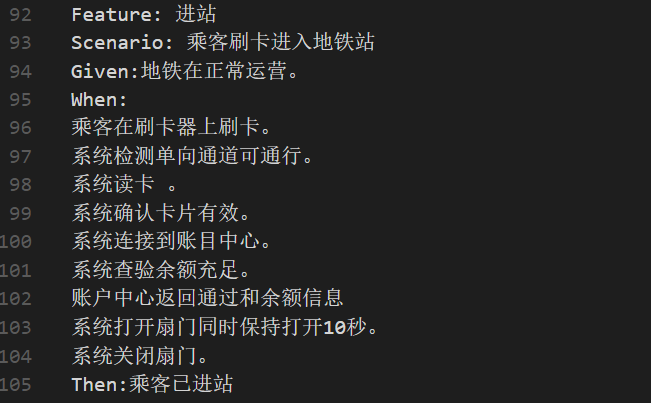


转化为RUCM：



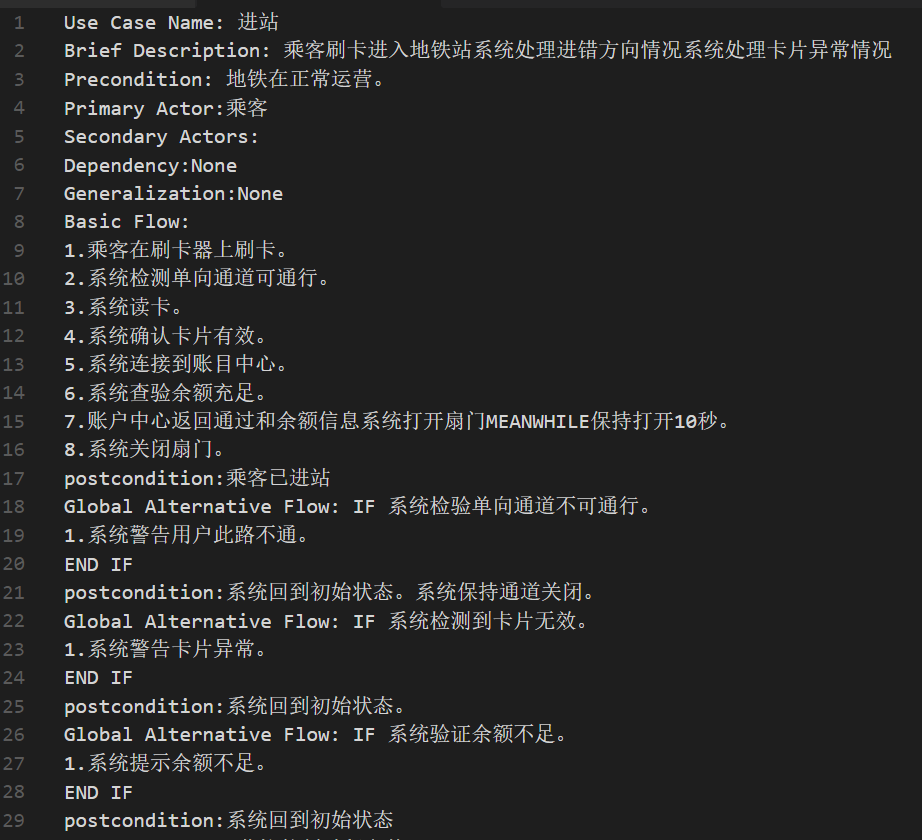


## 3.2 输入示例2：





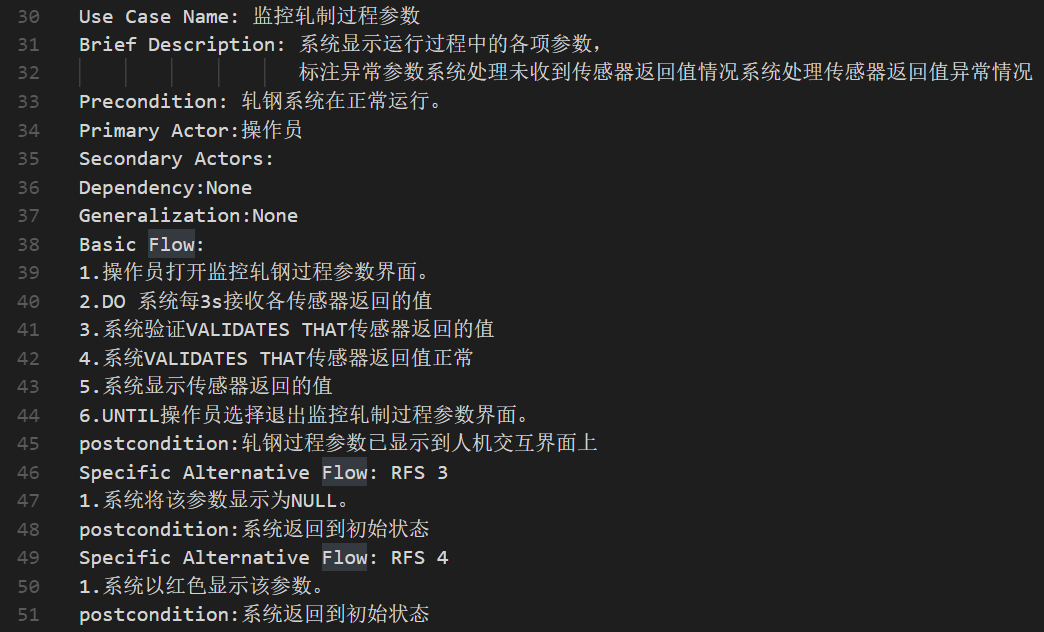
转化为RUCM：



## 3.3 输入示例3：



转化为RUCM：



## 4、界面展示



Browse：浏览本地文件

导入：将选择的文件传输至控制层

开始转换：触发控制层中的translation