



北京航空航天大學  
BEIHANG UNIVERSITY

BEIHANG UNIVERSITY

高等软件工程课程论文

## RUCMChecker

ZY1806707 石发强

ZY1806402 陈泽年

ZY1806705 梁保宇

ZY1806711 杨森

ZY1806501 黄子粤

@ 敏敏特穆尔队

Mentor 刘凯齐师兄

计算机学院

2018 年 12 月 30 日

## 目 录

1 项目概述 .....	3
1.1 软件设计架构 .....	3
1.2 软件输入输出样例 .....	3
1.2.1 输入样例 .....	4
1.2.2 输出样例 .....	4
2 项目设计 .....	6
2.1 类图说明 .....	6
2.1.1 规则解析部分 .....	6
2.1.2 规则实体部分 .....	9
2.1.3 报告部分 .....	10
2.1.4 RUCM 部分 .....	11
2.1.5 自然语言处理部分 .....	14
2.2 用户规则设计 .....	14
2.3 运行流程说明 .....	16
2.3.1 rucm 解析流程 .....	16
2.3.2 规则解析流程 .....	17
2.3.3 复杂规则检查流程 .....	17
2.3.4 gui 状态转移说明 .....	17
2.4 GUI 设计说明 .....	19
2.4.1 设计目的 .....	19
2.4.2 GUI 展示 .....	19
2.4.3 分模块说明与调用说明 .....	20
3 代码实现 .....	24
3.1 相关技术 .....	24
3.1.1 NLP .....	24
3.1.2 BFS .....	25

<b>目 录</b>	2
3.2 开发协作流程 .....	25
3.2.1 GIT .....	25
3.2.2 分模块开发 .....	26
3.3 问题与解决办法 .....	26
3.3.1 NLP Server .....	26
3.3.2 复杂规则的实现 .....	27
3.3.3 中文适配 .....	27
4 一致性验证 .....	29
4.1 OCL 控制 .....	29
4.2 第一次一致性检验 .....	29
4.2.1 代码与图之间的一致性 .....	29
4.2.2 时序图、活动图与类图之间的一致性 .....	29
4.2.3 时序图、活动图与代码之间的一致性 .....	29
4.3 第二次一致性检验 .....	29
4.3.1 代码与图之间的一致性以及各图之间的一致性 .....	30
4.3.2 设计与需求的一致性 .....	30
5 项目总结 .....	31
附录 A rule-template(规则模板) 说明 .....	32
附录 B 领域分析报告 .....	34
附录 C RUCM-Manual .....	64
附录 D 测试报告 .....	80

## 1 项目概述

RUCM 是一种结构化和模板化的需求规格，引入了流程、结构化句型和流程控制机制。本项目以 RUCM 编辑器产生的 rucm 文件作为输入，依据课堂所讲授的 RUCM 规范指定相应的规则，并按照规则来自动检查一个具体的需求违反了哪些规则，同时能够支持规则的设置。

关于 RUCM 模型的具体介绍请参见附录B部分领域分析报告。

### 1.1 软件设计架构

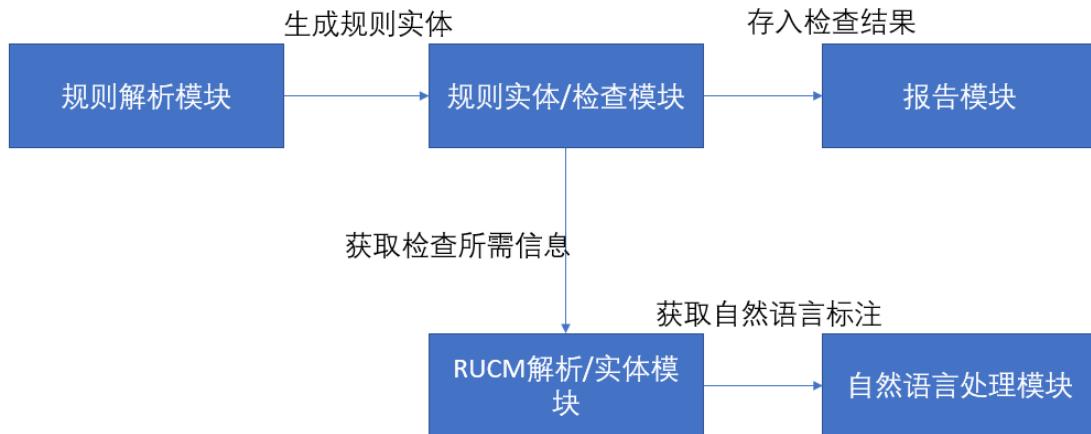


图 1.1 软件设计架构图

如图1.1所示，本软件由规则解析模块，规则实体/检查模块，报告模块，RUCM 接卸/实体模块，自然语言处理模块 5 部分组成。

其中规则解析模块负责规则格式解析域检查，规则存储等功能；规则实体/检查模块由 ComplexRule, SimpleRule, DefaultRuleXXX 等部分组成，该模块的主要功能是规则的表征与检查；规则报告模块负责存储报告信息以及生成报；RUCM 解析/实体模块负责检查/解析 RUCM 的 json 文件，并且将相应字段存入相应实体中并且提供对 step/sentence 等信息的统一查询接口；自然语言处理模块负责句子与词的内容分析以及相应标注。

### 1.2 软件输入输出样例

本软件以 RUCM 设计模型为输入，给出其是否符合 RUCM 规则的检查报告。

## 1.2.1 输入样例

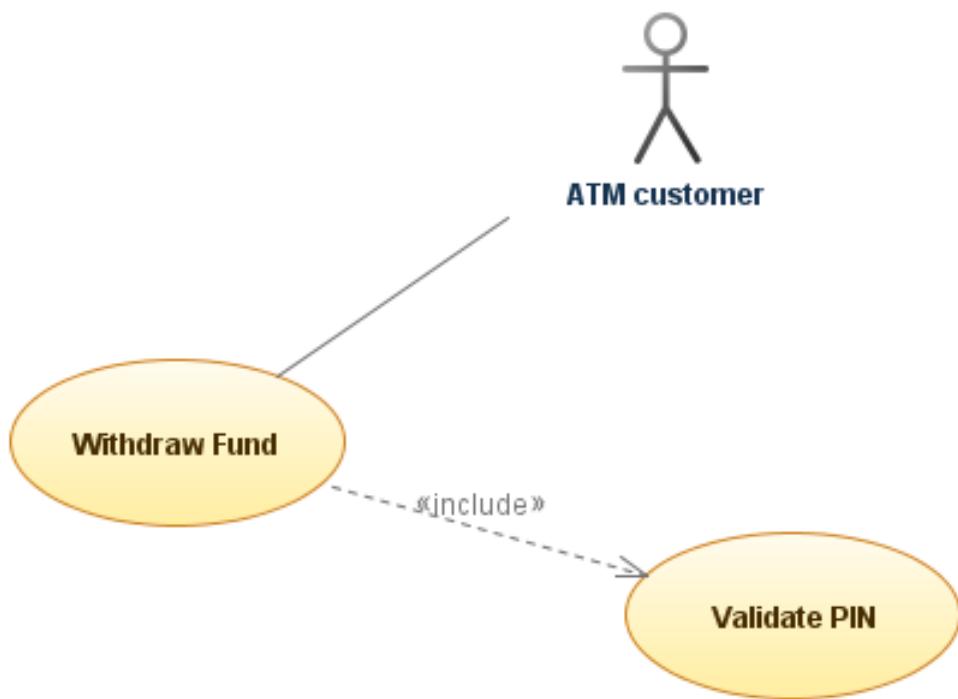


图 1.2 用例图样例

针对图1.2所示的用例图，我们写出了其用例描述1.3，其中红线标注部分明显违背了RUCM的第5、第10两条规则，即只能使用现在时的时态(R5)与不能使用情态动词(R10)。

关于RUCM规则的具体解释请参见附录C部分 RUCM-Manual 提供的官方解释。

## 1.2.2 输出样例

如图1.4所示，我们的软件检测到了相应的问题并输出了正确的报告。

R5	predicates.) Use present tense only.	multiple actions in a sentence. Enforce describing what the system does, rather than what it will do or what it has done.
R10	Don't use modal verbs (e.g., <i>might</i> )	Modal verbs and adverbs usually indicate uncertainty; therefore metrics should be used if possible.
R11	Avoid adverbs (e.g., <i>very</i> ).	
D12		
<b>Basic Flow</b> "Basic Flow" ▾		<b>Steps</b> 1 <b>INCLUDE USE CASE</b> Validate PIN 2 ATM customer selects Withdrawal through the system 3 ATM customer enters the withdrawal amount through the system. 4 ATM customer selects the account number through the system. 5 The system <b>VALIDATES THAT</b> the account number is valid. 6 The system <b>VALIDATES THAT</b> ATM customer has enough funds in the account. 7 The system <b>VALIDATES THAT</b> the withdrawal amount does not exceed the daily limit of the account. 8 The system <b>VALIDATES THAT</b> the ATM has enough funds. 9 The system dispenses the cash amount. 10 The system prints a receipt showing transaction number, transaction type, amount withdrawn, and account balance. 11 The system <b>might eject the card.</b> 12 The system displays Welcome message.  <b>Postcondition</b> ATM customer funds have been withdrawn.
<b>Bounded Alternative Flow</b> (Untitled) ▾		<b>RFS Basic Flow 5-7</b> 1 The system displays an apology message <b>MEANWHILE</b> the system ejects the ATM card. 2 The system shuts down. 3 <b>ABORT</b>  <b>Postcondition</b> ATM customer funds have not been withdrawn. The system is shut down.
<b>Global Alternative Flow</b> (Untitled) ▾		<b>IF</b> ATM customer enters Cancel <b>THEN</b> 1 The system cancels the transaction <b>MEANWHILE</b> the system ejects the card. 2 <b>ABORT</b> 3 <b>ENDIF</b>  <b>Postcondition</b> ATM customer funds have not been withdrawn. The system is idle. The system is displaying a Welcome message.
<b>Specific Alternative Flow</b> (Untitled) ▾		<b>RFS Basic Flow 8</b> 1 The system displays an apology message <b>MEANWHILE</b> the system ejects the ATM card. 2 <b>ABORT</b>  <b>Postcondition</b> ATM customer funds have not been withdrawn. The system is idle. The system is displaying a Welcome message.

图 1.3 RUCM 样例

```

Sentence ('{str': "The system displays Welcome message.", 'subjects': [the_system], 'verbs': [displays, Welcome], 'objects': [message], 'pronoun_count': 0, 'adverb_count': 0, 'modal_verb_count': 0, 'participle_count': 0, 'tense': diordfense.present('present')) in use caseWithdraw Fund violates the rule(default-4).
Sentence ('{str': "The system ejects the card.", 'subjects': [the_system], 'verbs': [eject], 'objects': [the_card], 'pronoun_count': 0, 'adverb_count': 0, 'modal_verb_count': 1, 'participle_count': 0, 'tense': diordfense.future('future')) in use caseWithdraw Fund violates the rule(default-5).
Sentence ('{str': "The system might eject the card.", 'subjects': [the_system], 'verbs': [eject], 'objects': [the_card], 'pronoun_count': 0, 'adverb_count': 0, 'modal_verb_count': 1, 'participle_count': 0, 'tense': diordfense.future('future')) in use caseWithdraw Fund violates the rule(default-10).
Sentence ('{str': "The system displays Welcome message.", 'subjects': [the_system], 'verbs': [displays, Welcome], 'objects': [message], 'pronoun_count': 0, 'adverb_count': 0, 'modal_verb_count': 0, 'participle_count': 0, 'tense': diordfense.present('present')) in use caseWithdraw Fund violates the rule(default-12).
Sentence ('{str': "ATM customer funds have been withdrawn.", 'subjects': [ATM_customer_funds], 'verbs': [have, been, withdrawn], 'objects': [], 'pronoun_count': 0, 'adverb_count': 0, 'modal_verb_count': 0, 'participle_count': 2, 'tense': diordfense.present('present')) in use caseWithdraw Fund violates the rule(default-12).
Sentence ('{str': "ATM customer funds have not been withdrawn.", 'subjects': [ATM_customer_funds], 'verbs': [have, been, withdrawn], 'objects': [], 'pronoun_count': 0, 'adverb_count': 0, 'modal_verb_count': 0, 'participle_count': 2, 'tense': diordfense.present('present')) in use caseWithdraw Fund violates the rule(default-12).
Sentence ('{str': "ATM customer funds have not been withdrawn.", 'subjects': [ATM_customer_funds], 'verbs': [have, been, withdrawn], 'objects': [], 'pronoun_count': 0, 'adverb_count': 0, 'modal_verb_count': 0, 'participle_count': 2, 'tense': diordfense.present('present')) in use caseWithdraw Fund violates the rule(default-12).
Sentence ('{str': "ATM customer funds have not been withdrawn.", 'subjects': [ATM_customer_funds], 'verbs': [have, been, withdrawn], 'objects': [], 'pronoun_count': 0, 'adverb_count': 0, 'modal_verb_count': 0, 'participle_count': 2, 'tense': diordfense.present('present')) in use caseWithdraw Fund violates the rule(default-12).
Sentence ('{str': "ATM customer funds have not been withdrawn.", 'subjects': [ATM_customer_funds], 'verbs': [have, been, withdrawn], 'objects': [], 'pronoun_count': 0, 'adverb_count': 0, 'modal_verb_count': 0, 'participle_count': 2, 'tense': diordfense.present('present')) in use caseWithdraw Fund violates the rule(default-12).

```

图 1.4 命令行形式下的软件输出样例图

## 2 项目设计

### 2.1 类图说明

我们的设计主要分为 6 个部分，分别是规则解析部分，规则实体/检查部分，报告部分，RUCM 解析部分自然语言处理部分以及图形界面部分，如图2.1。接下来分别说明每个部分重要的具体类图设计。

#### 2.1.1 规则解析部分

如图2.2所示，规则解析由类 RuleLoader 和 RuleDB 组成，RuleLoader 的作用是将规则文件解析成 ComplexRule/DefaultRule 等类，装入 ruleDB 中。

- RuleLoader:

`load()->bool`: 进行文件解析。文件解析结果将直接存放在 RuleDB 中。返回值代表是否解析成功

`checkFileFormat(rule:dict)->bool`: 检查文件格式是否符合要求，包含检查相应的字段是否存在，字段的值是否合法，各个字段之间的关系是否符合约束

`parseDefaultRule(rule:dict)`: 解析默认规则

`parseComplexRule(rule:dict)`: 解析复合规则字典，具体规则格式详见 rule-template.md

`parseSimpleRule(rule:dict)`: 解析简单规则

`checkComplexRule(rule:dict)`: 检查复合规则格式

`checkSimpleRule(rule:dict)`: 检查简单规则格式

- RuleDB:

ruleDB 为静态数据类，它根据两部分组成，分别是用户定义规则列表 (`userRules:list`) 和默认规则列表 (`userRules:list`) 组成

规则解析部分 ocl:

`context: Loader`

`inv: self.filepath <> None`

`context: RucmLoader::load()`

`postCondition: RUCMRoooot.usecases.size() >0 and RUCMRoooot.actors.size() >0`

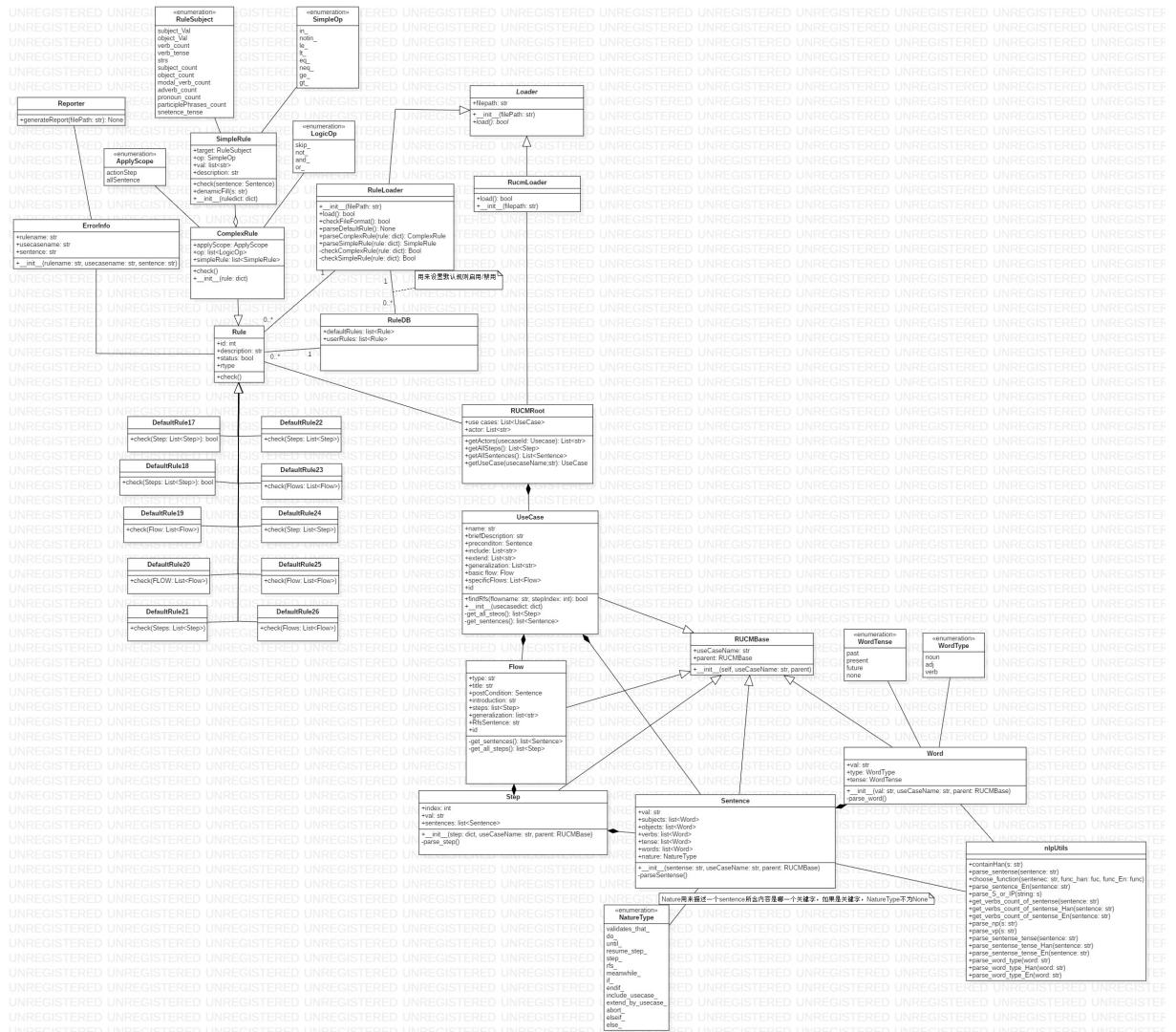


图 2.1 主体部分类图概览

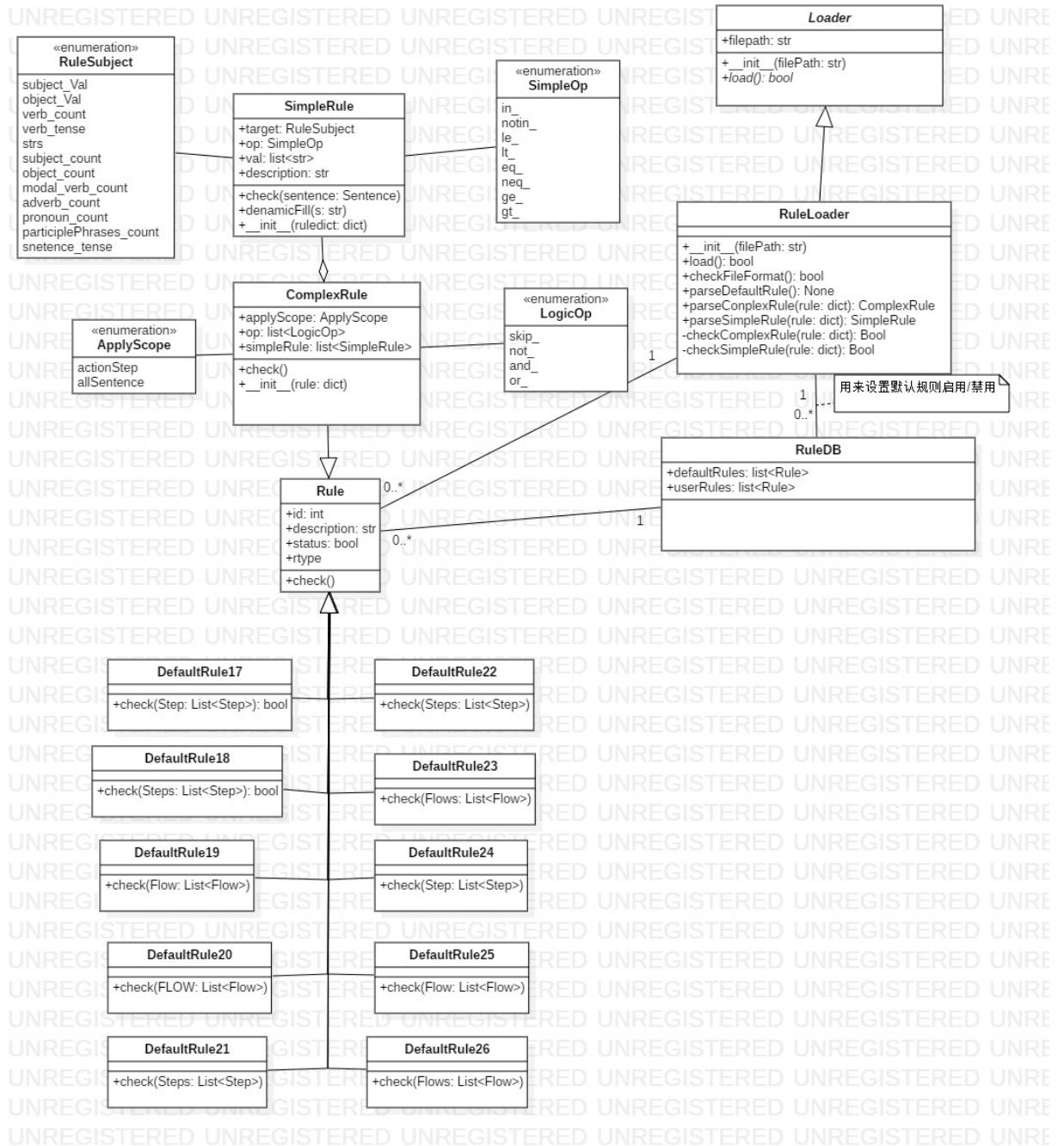


图 2.2 规则解析与规则实体部分类图

context: RuleDB

inv: self.defaultRules <> None and self.userRules <> None

### 2.1.2 规则实体部分

规则实体部分如图2.2所示。主要分为 defaultRule, Rule, ComplexRule 三大类。defaultRule 实现关键字规则的检查, Rule 是所有规则类的基类。ComplexRule 负责所有形式化规则检查。

- Rule: 所有的规则都继承自 rule 基类, Rule 的子类包括 DefaultRuleXXX, ComplexRule (SimpleRule 不是 Rule 的子类)。

id:rule 规则表示

description: 规则描述

status: 是否启用

rtype: 规则的类表, rtype 必须取'user','system' 之一

- ComplexRule:

一个复合规则可以由多个简单规则的检查结果综合而成

applyScope: 规则的作用域, 目前作用于可以为 rucm 中的 action 字段或者所有的句子。

simpleRule: 简单规则列表

op: 对简单规则的综合逻辑操作

- SimpleRule:

一个 simpleRule 只检查一个句子中的一个字段, 它的检查方式可以抽象为某个句子成分在/不在目标列表中。

target: 要检查的句子成分, 具体取值可以参见 rule-template.md

op: 最检查目标的逻辑约束可以是 in/not in

val: 允许/禁止列表

check(sentence: Sentence): 检查句子

dynamicFill:(s:str): 动态填充 val, 比如填充 rucm 中的 actor

DefaultRuleXXX:

无法被抽象成 ComplexRule 形式的规

规则实体部分 ocl:



图 2.3 报告部分类图

context: Rule

inv: self.id >= 0 and self.description <> None and collection("user", "system").includes(self.rtype)

context: ComplexRule

inv: self.op <> None self.op.size() >0 and self.simpleRule.size() >0

context: SimpleRule

inv: self.val <> None and self.description <> None and self.val.size()>0 and (if(collection(subject\_count, ob) then self.val.forAll(x|type(x) == Integer) else true) and (if collection(ge, gt, eq, neq, lt, le).include(self.op) then self.val.forAll(x|type(x) == Integer else true))

context: SimpleRule::dynamicFill()

postCondition: return = self.val.forAll(s:str|s.contains("\$")) = Flase)

### 2.1.3 报告部分

- ErrorInfo:

包含检查错误结果的基本信息，包括检查出错误的规则，相应的用例名以及句子。

- Reporter:

静态类，用于生成报告。

报告部分 ocl:

context: ErrorInfo

inv: self.rulename <> None and self.usecasename <> None and self.sentence <> None

```
context:Reporter  
inv:self.errors <> None
```

### 2.1.4 RUCM 部分

我们为了方便规则类进行相应信息的获取，设置了 RUCMRoot 类对 usecase 禁停统一管理，并且提供了对于 sentence, Action 等信息的痛惜查询接口。根据 rucm 文件结构与逻辑组织方式，我们从顶层到底层分别设置了 Usecase, Flow, Action, Sentence, Word 类。如此进行设置不仅方便 RUCM 的解析，也简化了后续的检查流程。

- RUCMRoot:

存储所有的 RUCM 信息包括 actor 列表域 usecase 列表，并且提供相应 get 方法方便 rule 查询信息

usecases: 用例列表

actors: actor 列表

getActors(): 获得 actor 列表

getAllSteps(): 获得所有 step 列表

getAllSentences(): 获得所有的句子

getUseCase(usecasename:str): 查找相应的 usecase

- Usecase:

对应一个用例

id: 用例 id

name:usecase 的名字

briefDescription:usecase 简单描述

preCondition: 前置条件

include: 该用例所包含的用例

extend: 该用例 extend 的用例

generalization: 该用例泛化的用例

basicFlow: 用例的基本流

specificFlows: 用例分支流

findRfs(flowname:str,stepIndex:int)->bool: 判断名字叫 flowname 的 flow 中是否存在序号 stepIndex 的步骤

- Flow:

type: 区分 basicflow/Specific Flow/Global Alternative Flow

name: flow 名称

postcondition: flow 的 postcondition

steps: 构成 flow 的 step

RfsStatement:specific flow 的 RFS 字段

id:flow 的 id

- step:

一个 step 可以由多个 sentence 构成

index: step 的序号

val: step 的字符串

sentences: step 字句

parse\_step(): 解析句子

- sentence:

一个 sentence 可以是一个正常的自然语言句子，也可以是一个关键字（IF/ELSE 等）

val: sentence 字符串

nature: sentence 的关键字类别

- RUCMBase:

所有的 RUCM 元素的基类，提供向上查找父节点以及所属 Usecase 的相应属性

word:

val: 词的字符串

type: 词的类别（noun/adj/verb 等）

tense: 词的时态（past/present 等）

rucm 部分 ocl:

context:RucmLoader::load()

postCondition: RUCMRoooot.usecases.size() >0 and RUCMRoooot.actors.size() >0

context: Word

inv: self.val <> None and self.type <> None and self.tense <> None

context: Sentence

inv: self.val <> None and self.subjects <> None and self.objects <> None and self.verbs <> None and self.tense <> None and self.words <> None

context: Step

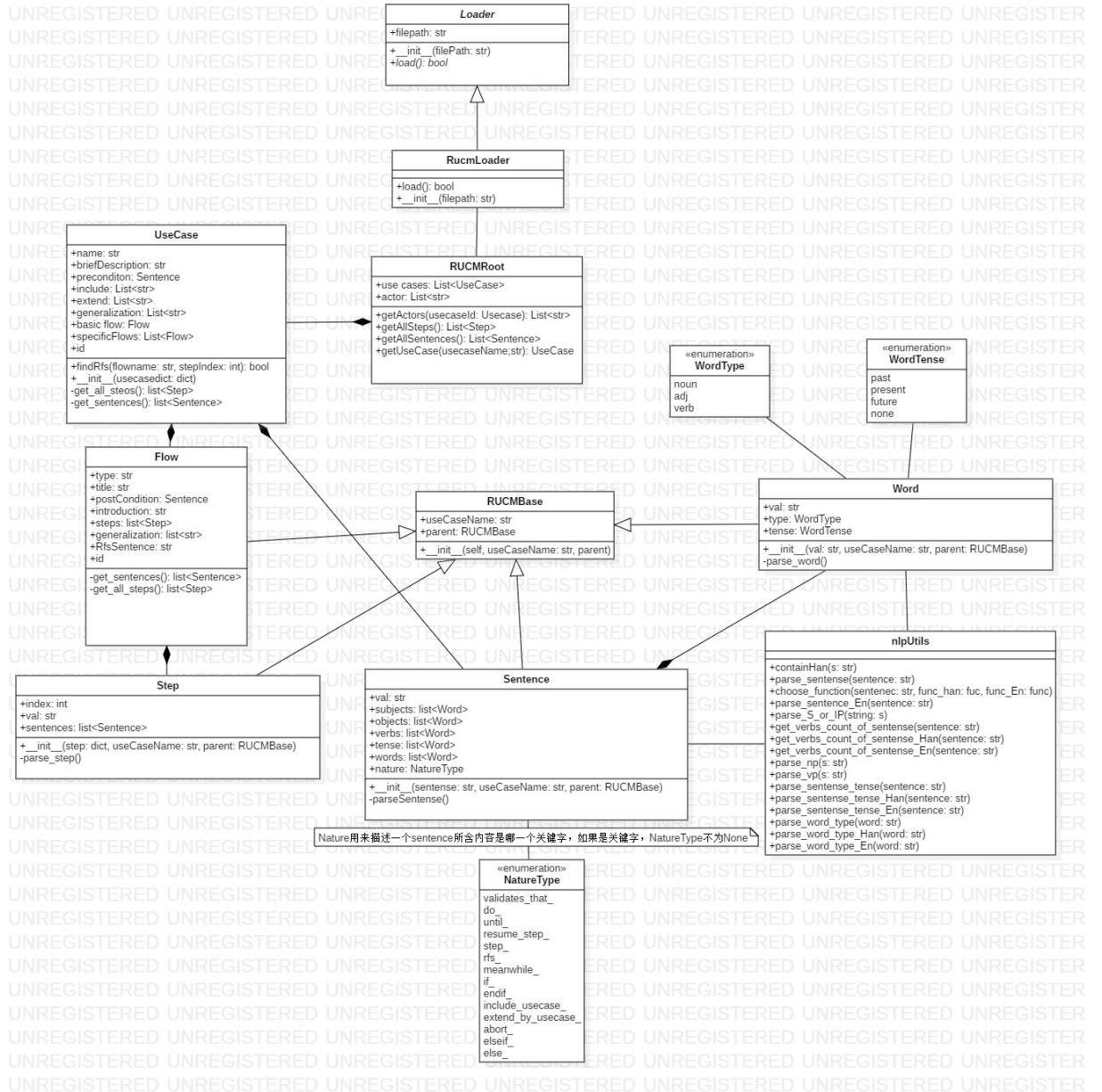


图 2.4 报告部分类图部分类图

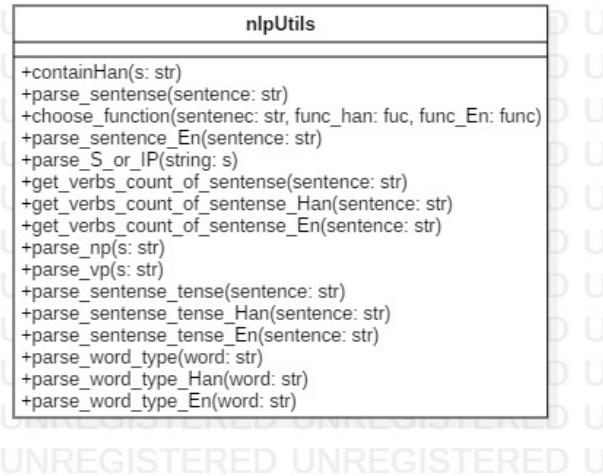


图 2.5 自然语言处理部分类图

```

inv: self.index >= 0 and self.val <> None and self.sentences <> None and self.sentences.size()
>0

context: Flow

inv: collection("BasicFlow", "Specific Flow", "Global Alternative Flow").includes(self.type) and
self.title <> None and self.postCondition <> None and self.introduction <> None and self.steps
<> None and self.steps.size() >0 and self.include <> None and self.extend <> None and self.generalization
<> None and self.rfsSentence <> None

context: UseCase

inv: self.name <> None and self.briefDescription <> None and self.postCondition <> None and
self.include <> None and self.extend <> None and self.generalization <> None and self.basicFlow
<> None and self.specificFlows <> None and self.id >= 0

```

### 2.1.5 自然语言处理部分

如图2.5所示，该部分由多个方法组成，构成 nulputils 模块，该部分的主要功能是解析句子/词语成分，并且给予相应的自然语义标注。因为中文的语法分析与英文的语法分析有较大不同，我们设置 XXX\_En 和 XXX\_Han 对其进行分别处理。并且设置方法 choose\_function 对语言进行选择

## 2.2 用户规则设计

rule 文件有一个 dict 组成，其中有两个字段，分别是 default 和 user-def。分别代表默认规则列表和用户定义规则列表。

default 字段下的数据尽量不要改，唯一可以改的是 status 字段，当 status 为 false 时，规则处于禁用状态

用户规则列表由用户规则聚合而成，一个用户规则下的字段包括：

- id: 用户规则 id，不能重复
- applyScope: 规则作用域，取值包括”actionStep”(flow 中的句子), ”allSentence”(RUCM 中所有句子)
- simpleRules: 一个用户规则可以被看做多个简单规则的聚合，简单规则字典信息包括：
  - subject: 规则要检查的对象，他的取值包括：
  - subject\_Val: 主语的值
  - subject\_count: 主语的数量
  - object\_Val: 宾语的值
  - object\_count: 宾语的数量
  - verb\_count: 动词的数量
  - verb\_tense: 动词的时态
  - strs: 句子中所有的词
  - modal\_verb\_count: 情态动词的数量
  - adverb\_count: 副词的数量
  - pronoun\_count: 代词的数量
- operation: 对检查对象的约束，可以是 in/notin/le/lt/eq/neq/ge/gt，分别表示约束对象在/不在/小于等于/小于/等于/不等于/大于等于/大于
- val: 约束列表，例如 ”subject”: ”subject\_Val”, ”operation”: ”in”, ”val”:[”system”]‘表示主语中不能出现 system 字眼。列表中除了出现正常字符串以外，还可以出现”\$actor”代表 RUCM 中的 actor
- 注意：当 subject 字段为 XX\_count 的时候，val 字段中只能出现数字，当 operation 为除了 in notin 以外的字段时候，val 字段中只能出现数字。
- status: 当 status 为 false 时，规则处于禁用状态
- operation: 综合简单规则的逻辑操作
- 第一个字符代表对整个检查结果的操作”-” 代表无操作，”!” 代表取非操作

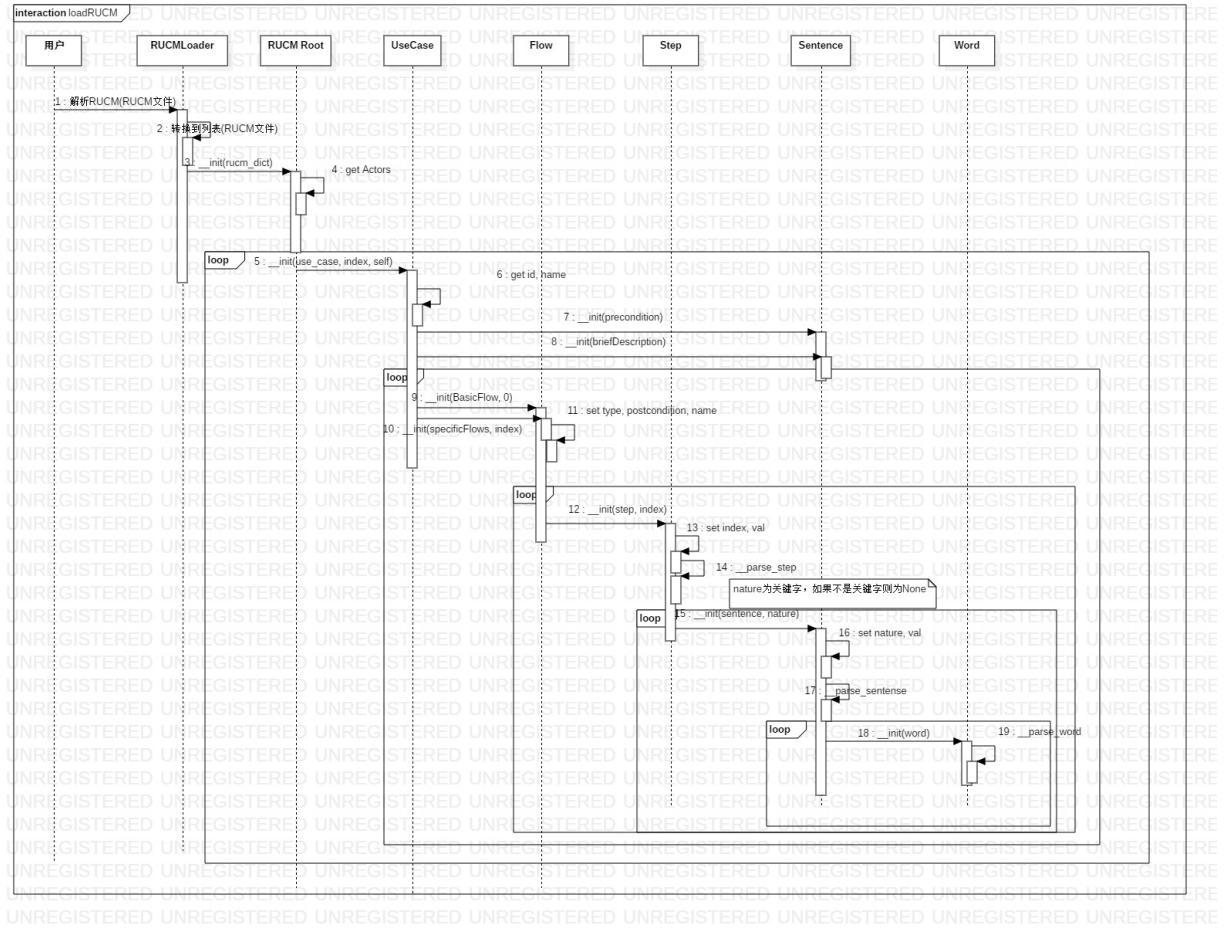


图 2.6 rucm 解析流程图

- 第一个字符往后分别代表对于简单规则之间的逻辑操作，”&” 代表与操作，“|” 代表或操作
- description: 规则描述
- 注意：通过规则模板编辑方式可以编写内含多个条件约束的规则，而通过 GUI 编辑方式获得的规则只能包含一个条件约束。

## 2.3 运行流程说明

### 2.3.1 rucm 解析流程

rucm 解析流程如图2.6所示，rucmLoader 通过调用 RUCMRoot 的 init 方法生成 rucm 相关类的实例。具体来说，rucm 的解析是 useCase,Flow,Step, Sentence, word 等类依次反复迭代调用的过程。在生成 Sentence 和 Word 的过程中，sentence 和 word 会调用自然语言处理模块获得相应的语法标注。

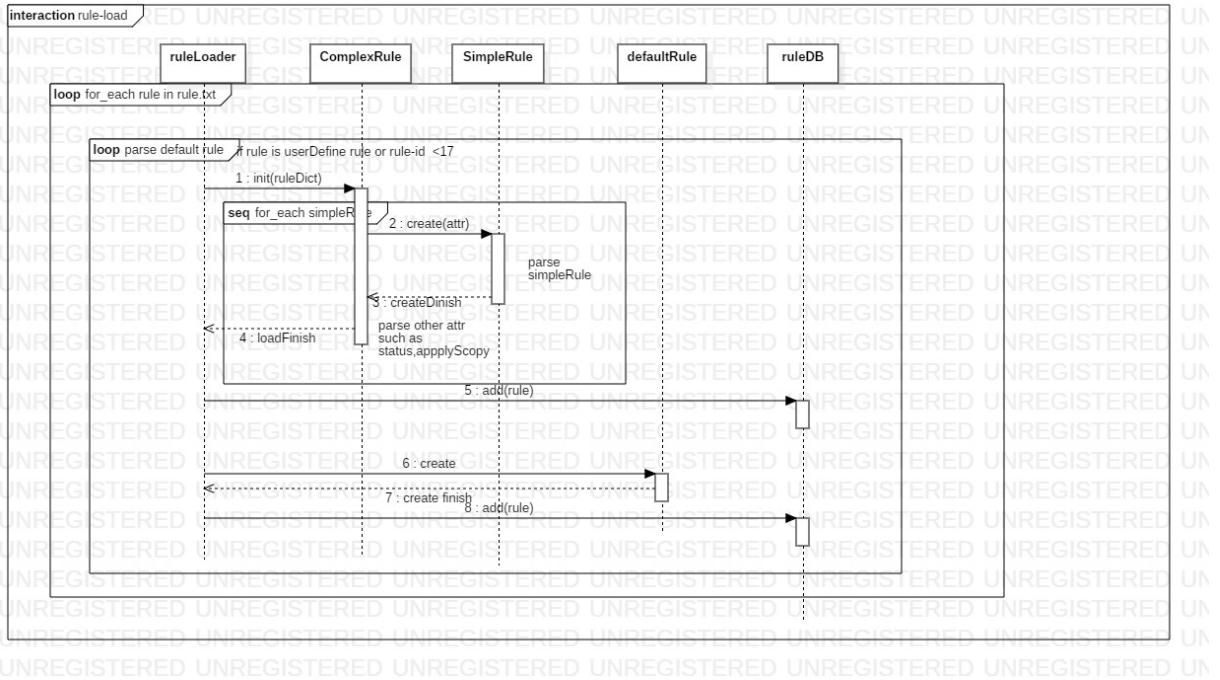


图 2.7 规则装载流程

### 2.3.2 规则解析流程

规则解析流程如图2.7所示。规则解析分为两个部分，默认关键字规则解析和复杂规则解析，默认规则解析就是把对应 id 的规则实例化。而复杂规则的解析分为解析相应的简单子规则和解析复杂规则相应字段连个部分。当规则解析完毕后，规则被添加到了规则数据库中。

### 2.3.3 复杂规则检查流程

复杂规则解析流程如图2.8所示。一个复杂规则可以由多个简单规则的检查结果综合而成。在进行复杂规则检查的时候，复杂规则先对下属的简单规则进行检查，待检查完成的时候将简单规则的检查结果进行综合获得最终检查结果。因为有些规则的约束值因 rucm 的文件不同而不同（如对 actor 的约束就是因 rucm 文件变化而变化的）

### 2.3.4 gui 状态转移说明

gui 状态转移如图2.9所示。图形界面主要状态有三个：加载规则状态，就绪-rucm 文件未设置状态和就绪 rucm 文件设置完成状态。两个就绪状态分别可以迁移到查看规则和添加/修改规则状态。通过设置 rucm 文件可以将“就绪-rucm 文件未设置”状态迁移到“就绪-rucm 文件设置完成”状态。当检查完成后，状态机返回“就绪-rucm 文件未设置”

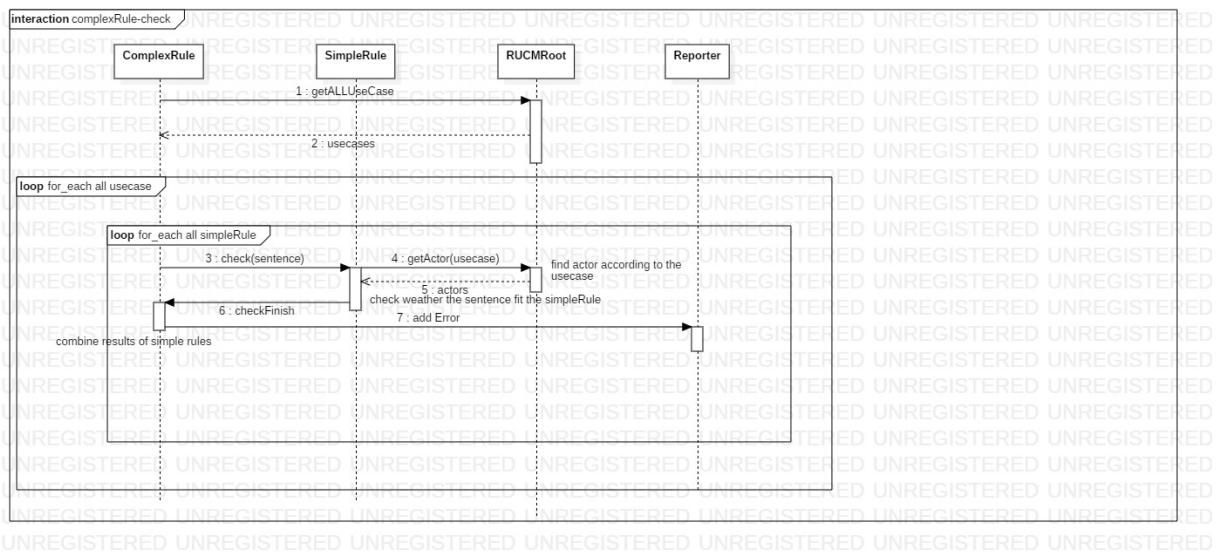


图 2.8 rucm 解析流程图

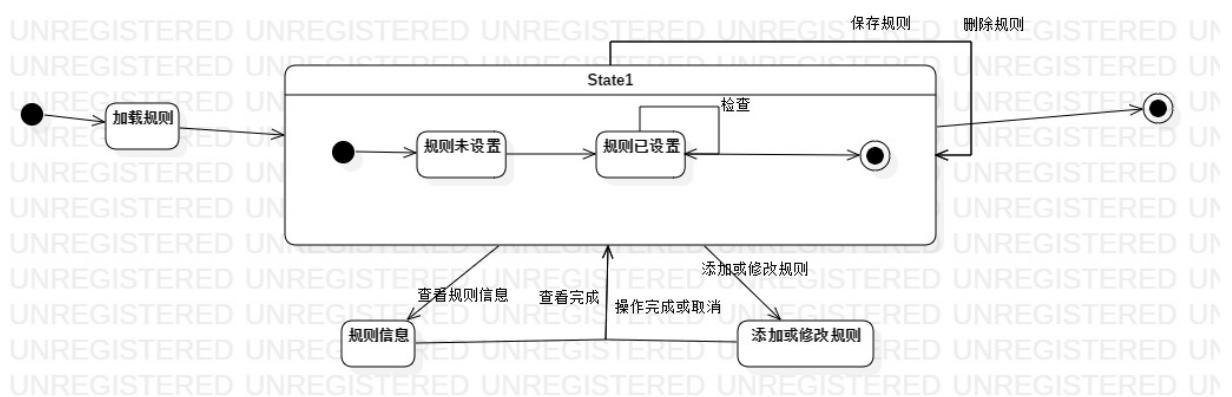


图 2.9 图形界面状态转移图

## 状态

### 2.4 GUI 设计说明

#### 2.4.1 设计目的

利用图形界面，为用户提供可视化的输入/输出与控制功能，包括 RUCM 规则编辑功能、规则存取功能、RUCM 文件读取功能、规则检查功能和查看报告界面，方便用户理解与使用。

#### 2.4.2 GUI 展示



图 2.10 RUCMChecker 主界面

软件的主界面如图2.10所示。该界面分别提供了如下功能。

- 1、 默认规则库相关功能。默认规则库相关功能包括默认规则的查看、默认规则的启用与禁用设置。
- 2、 用户规则库相关功能。用户规则库相关功能包括从文件中载入用户规则、保存用户规则到文件、添加用户规则、查看用户规则列表、查看用户规则、修改用户规则以及删除用户规则。
- 3、 RUCM 文件相关功能。该模块主要包括从文件系统中载入 RUCM 文件。
- 4、 控制相关功能。包括规则检查与查看规则检查报告。

### 2.4.3 分模块说明与调用说明

对应主界面提供的功能，RUCM 规则检查器的图形化窗口设计共包括四个模块：默认规则模块、用户规则模块、RUCM 文件模块以及控制模块。

#### 默认规则模块

默认规则模块在程序运行伊始自动载入默认规则，并在主界面中“默认规则库”表格中显示系统中的默认规则。默认规则模块为用户提供了启用/禁用的控制权限，用户可以通过选择启用/禁用若干默认规则，控制在规则检查中是否使用这些默认规则进行检查。

#### 用户规则模块

用户规则模块为用户提供了规则编辑的接口。在程序运行伊始，用户规则模块默认为空。用户可以通过点击“添加用户规则”按钮添加用户规则。添加用户规则界面如图2.11所示。其中，作用范围、作用对象、操作的可选项详见本文档规则设计部分。在添

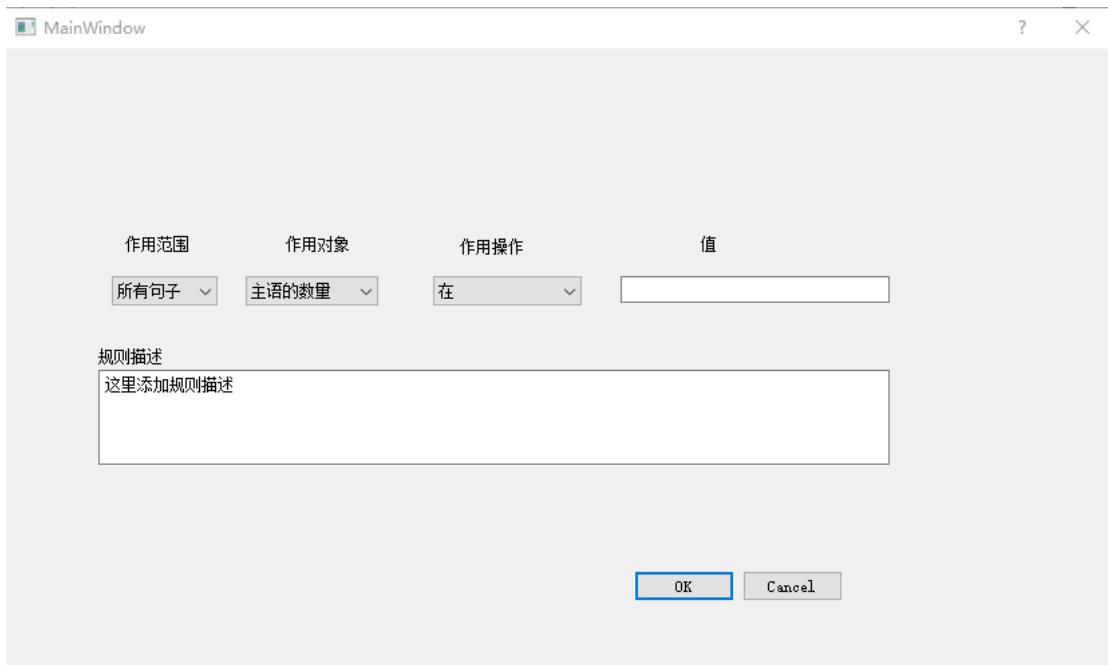


图 2.11 添加用户规则界面

加一条用户规则后，主页面的用户规则库中将添加该用户规则。此外，用户可以通过载入规则库/保存规则库进行用户规则的载入与保存。在载入用户规则文件后，系统在主页面中的用户规则库中填入载入的规则。载入/保存界面如图2.13所示。对于单条规则，



图 2.12 更新添加的用户规则

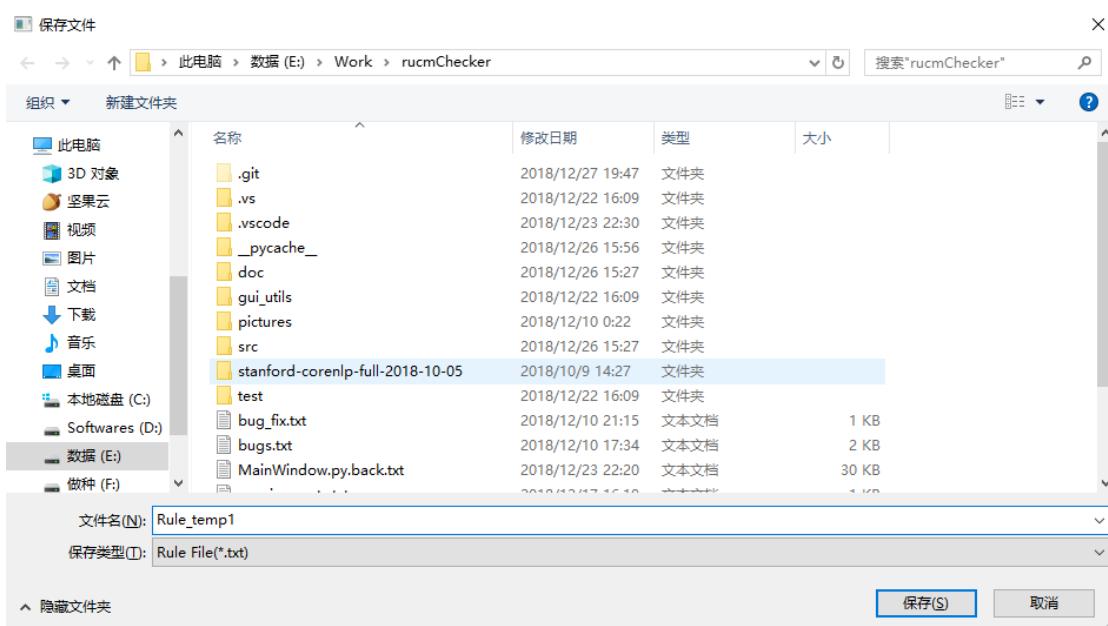


图 2.13 载入/保存用户规则界面

软件的图形化界面提供了查看、修改与删除的操作。

查看规则 用于显示规则的详细信息，其界面如图2.14所示。

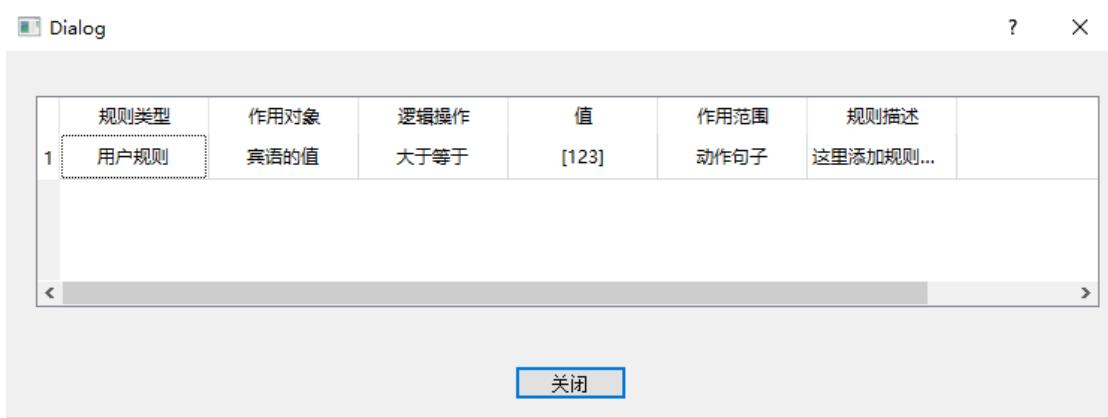


图 2.14 用户规则查看界面

修改规则 用于更新规则的详细信息，其界面如图2.15所示。

删除规则 用户删除某条用户规则。

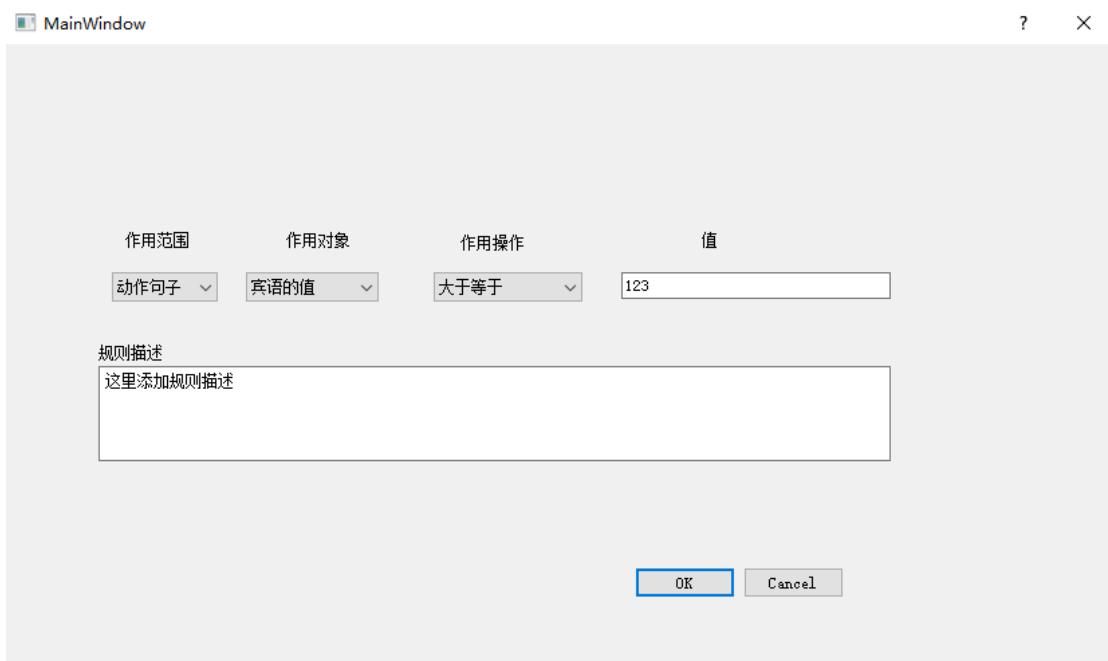


图 2.15 规则修改界面

## RUCM 文件模块

该模块为用户提供了上传 RUCM 文件的接口。在用户点击选择文件后，程序调出文件上传界面，在上传界面选择文件后，主界面的 RUCM 输入框显示上传的 RUCM 文件地址，如图2.16所示。

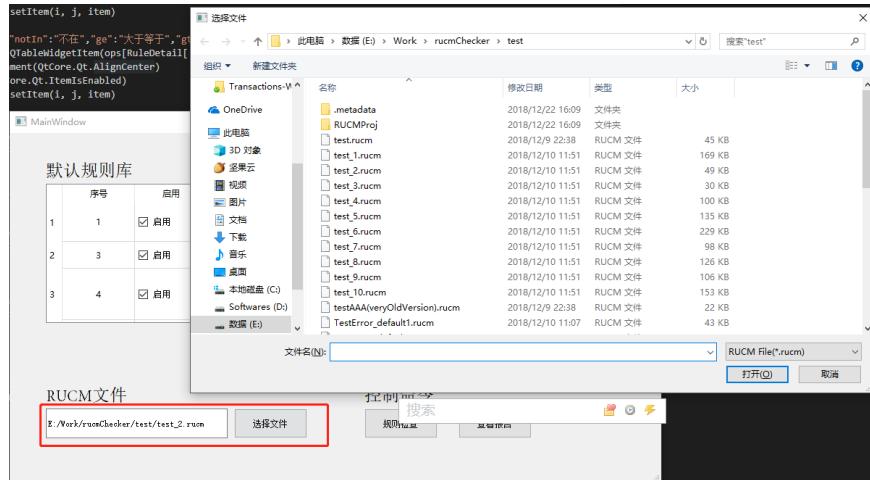


图 2.16 RUCM 文件上传界面

## 控制模块

该模块为用户提供了规则检查与查看报告的接口。在用户点击“规则检查”后，程序根据规则库的内容以及 RUCM 文件地址，调用后端程序进行检查，并在主界面下方显示“正在检查”的提示。为确保程序能够及时响应用户操作，在 UI 中采用多线程的方式调用后端程序。规则检查完毕后，主界面下方显示“检查完毕，请点击查看报告”。用户点击查看报告后，出现查看报告界面，界面内容如图2.17所示。

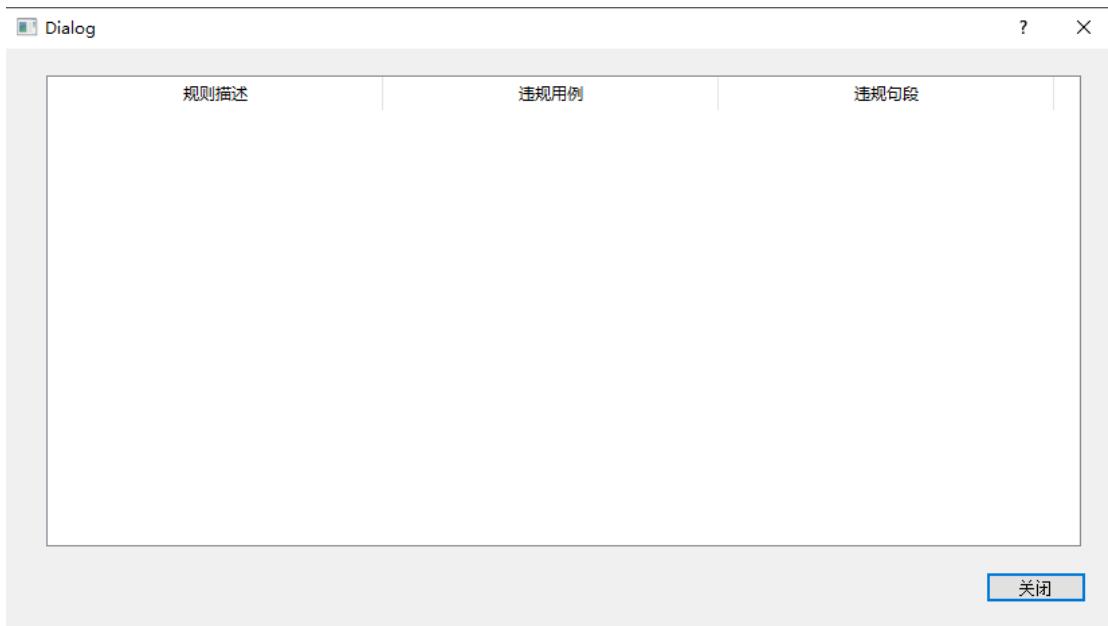


图 2.17 查看报告界面

### 3 代码实现

#### 3.1 相关技术

##### 3.1.1 NLP

由于规则检查中设计到大量的自然语言处理的部分，我们将其独立开发为一个模块 `nlputils.py`。其对外提供了所有关于自然语言处理的接口。

包括：

`parse_sentence(sentence)`, 输入参数为一个类型为字符串的句子，输出(返回值)为3个字符串的列表，分别表示句子中的主语、谓语动词和宾语。

`get_verbs_count_of_sentence(sentence)`, 输入参数为一个类型为字符串的句子，输出(返回值)为4个字符串的列表，分别表示句子中的代词、副词、情态动词、分词。

`parse_sentence_tense(sentence)`, 输入参数为一个类型为字符串的句子，输出(返回值)为一个字符串，表示该句子的时态：

`present` 现在时

`past` 过去时

`future` 将来时

`none` 其他

`parse_word_tense(sentence)`, 输入参数为一个类型为字符串的单词，输出(返回值)为一个字符串，表示该单词的时态：

`present` 现在时

`past` 过去时

`future` 将来时

`none` 其他

`parse_word_type(sentence)`, 输入参数为一个类型为字符串的单词，输出(返回值)为一个字符串，表示该单词的类型：

`verb` 动词

`noun` 名词

`adj` 形容词

`none` 其他

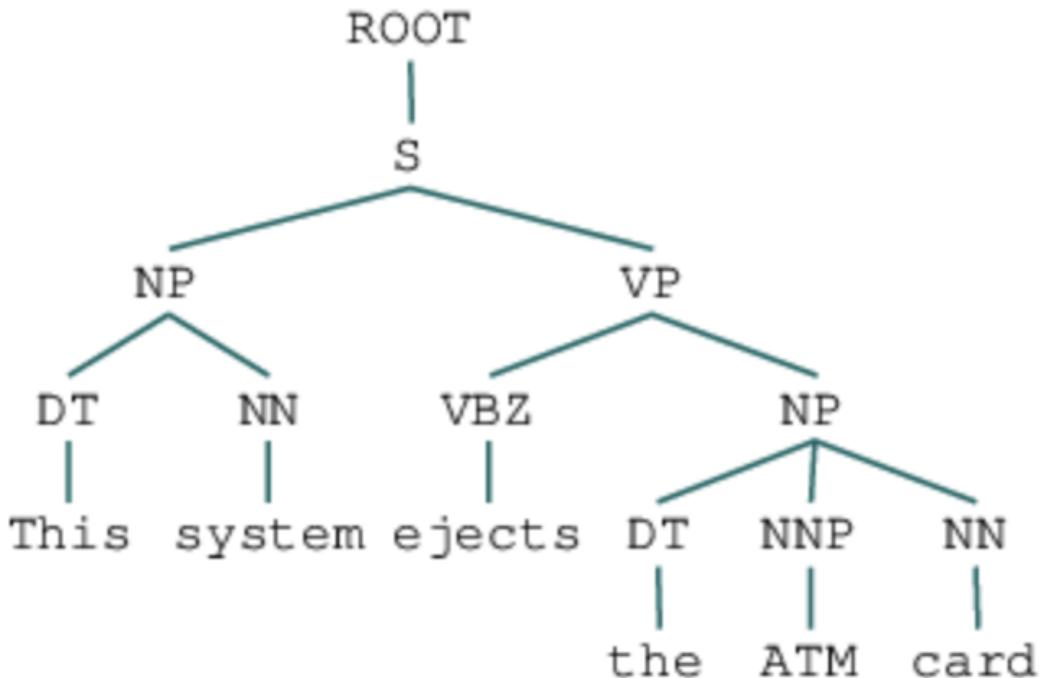


图 3.1 语义分析语法树

### 3.1.2 BFS

在处理句子时，使用 nlp 工具可以获得句子的每个单词的成分，句子的构成就成了一棵树，如图3.1所示：

为了遍历到每个单词，我们采用广度优先搜索算法进行遍历。因为这样可以保证相同层的节点被顺序遍历到。

## 3.2 开发协作流程

### 3.2.1 GIT

本项目的开发协作使用 git 进行版本控制，并将 repo 放在 GitHub 上完全开源：*rucm-Checker*。

开发主要以 2 个分支的形式进行，一个 master，一个 lby。主要功能在 master 上开发，检查规则的功能在 lby 上开发，为了避免最后合并时冲突过多，lby 分支在开发过程中会不断 merge master 上的更新。最后在合并分支时，我们选择集体编程这一模式。所有成员坐在一起，快速处理合并带来的冲突和 bug，在 2 小时内完成合并任务，获得了可以正确运行的代码。在之后的 bug 修复、功能添加和测试时，所有的成员都在 master 上工作，不再使用新的分支。所有的 commit 和分支的历史信息，都可以在 GitHub 上找

到，方便我们控制代码的所有版本，在必要时进行回退。并且知道每一行代码的修改者，在遇到 bug 或疑问时可以快速找到源头，进而解决。

### 3.2.2 分模块开发

根据设计，我们将所有的代码分为 2 大块：

- 1、项目核心的规则检查部分
- 2、其它辅助部分，包括规则和 rucm 的加载，错误报告，参数处理，GUI 交互等内容

实际开发过程中，首先，架构师同学先根据类图写好框架，把所有需要的类定义出来，其相应的方法和属性也都写好，参数含义、类型，返回值含义、类型也都通过 typing 包或者名字、注释规定清楚。然后，程序员再开始分为 2 波，各自负责一块代码。当所有模块都开发完成后，所有人聚在一起，合并模块和代码，处理冲突和 bug。事实证明，这种合作开发方式是十分高效和舒适的。大家各司其职，减少了大量开会和讨论的时间。合并完成后，测试人员进行测试，发现 bug 后，通过 git 的 commit 信息追踪到产生 bug 的程序员，将修复 bug 的任务交付给他。

## 3.3 问题与解决办法

### 3.3.1 NLP Server

nlp 函数都依赖于 nltk 库和一个用于自然语言处理的 Java server。开发过程中，最开始我们启动本地的 server 完成了代码。但在不同程序员完成各自模块，大家开始合并的时候，发现这样的设计并不友好。一方面，有些程序员并不能在本地顺利地配置好 Java 环境，启动本地 server；另一方面，如果最后展示的时候程序还要依赖本地的 server 的话，还要给用户多加一步配置 Java 环境，启动 server 的步骤。经过讨论和商议，改为了在校园网的一台机器上启动这个 server，24 小时监听，程序默认会访问这个 server，同时，给出一个参数--url，即 server 的链接。允许用户自己配置 server。这在对于校园网外的用户和需要自己内部使用 server 的用户来说也是可以接受的。而在最后的 GUI 版本中，我们经过一段时间的试用基本确定了校园网内通过网络服务器提供 NLP 处理的稳定性与可用性，因此对用户屏蔽了这些负责的设置内容，直接配置为固定的远程 NLP 服务。

### 3.3.2 复杂规则的实现

在实现和测试过程中，我们发现原有的设计并不能完全实现所有的规则检查。对于一些可以实现的规则，我们选择了修改原有设计，比如，在最开始的设计中，句子这个类并没有代词数量这一属性，但只有有了这个属性，一条规则才能很方便的实现。出现这种情况的原因在于，设计时虽然尽可能考虑所有的规则和情况，但毕竟抽象程度还是高于实现的，所有有些设计并不与实现完全对等。由于原来设计的低耦合性，在修改设计后，代码的修改和更新是十分迅速的。对于另外一些无法实现或没有必要实现的规则，我们选择在说明书中会陈述清楚，暂时不支持这些规则的检查。这样可以快速地生成可以交付给用户使用的版本，对于一些瑕疵选择之后的版本再解决，更加符合敏捷开发的原则。

### 3.3.3 中文适配

在第一次的展示中，吴老师给出要增加可以处理 rucm 中出现中文的情况，并检查其他组的 rucm 文件作为目标。增加对中文的适配，难度主要有 2 点：

- RUCM manual 中，所有 26 条规则的定义和示例都是针对英文的，完全没有考虑中文。比如，对于第 15 条规则，中文中并不存在分词。所以我们需要重新定义 26 条默认规则，删除不必要和不方便实现的默认规则。
- 中文 NLP 技术与英文的差别还是很大的，尤其是语法分析部分，句子成分和词性区别很大。很多函数需要准备 2 套规则，分别适应于不同的语言。对于一个句子中可能同时出现中英文的情况，比如有些同学的 rucm 文件的关键字和名词用英文，其他地方使用中文，对于这种情况，我们经过讨论决定，不予支持。即不允许中英文在一个句子里混杂，但是允许不同句子使用不同的语言。

中文 NLP 主要借助了 2 个工具：结巴分词和 Standford parser for Chinese。其中，结巴分词主要用于，对中文进行分词，和词性分析。Standford parser for Chinese 用于中文的语法分析。工具的选择主要考虑到开发速度、易用性和准确性。英文中不存在分词问题，每个单词都是以空格分隔的。但中文中，分词确实被广泛研究的，是一切 NLP 的基础。经过调研，“结巴分词”使用率最广。我们选择使用“结巴分词”作为我们的分词工具和词性分析工具。结巴分词并不提供更高级的 NLP 技术。对于语法分析，我们仍然选择 Standford 的 parser。主要考虑是，Standford parser for Chinese 的接口和使用方法与 Standford parser for English 相同。直接使用它可以降低学习成本，提高开发效率。

针对可能出现的中英文混杂，我们实现了自动切换分析工具的功能。即根据句子的

语言类别，使用 2 套不同的函数进行分析、NLP 处理。降低了用户使用的难度，提高了可用性。

## 4 一致性验证

一致性验证的任务是检查各图之间的名称、关系的一致性以及图与代码之间的一致性。

### 4.1 OCL 控制

在本项目设计与开发的过程中，我们引入了 OCL 方式描述的 UML 模型的一致性规则，并在开发的过程中通过遵循 OCL 控制的方式来保证设计与实现过程中的一致性。在完成开发后，我们再一次分阶段进行了全面的一致性检验工作。

### 4.2 第一次一致性检验

#### 4.2.1 代码与图之间的一致性

不一致的类	不一致情况频率	类图修改方案
Flow	6	删除了 include、generation、extend 方法及属性，在实现中没有用到
Step	1	添加了 parse_step 私有方法
Sentence	1	添加了 parseSentence 私有方法
NatureYype	2	添加了 else_，删除了重复属性
RUCMLoade	5	修改了返回值，添加了两个私有方法
RuleSubject	1	删除了两个属性，在实现中没有用到

#### 4.2.2 时序图、活动图与类图之间的一致性

除部分名称从类图中的英文转变为时序图中的中文外，并未发现不一致性。

#### 4.2.3 时序图、活动图与代码之间的一致性

各个图与代码之间基本符合，但是时序图相较于代码过于简略，没有很好的呈现出代码的流程。

### 4.3 第二次一致性检验

在经过第一答辩评审后，结合中文适配、GUI 界面设计、用户规则抽象等新内容的引入，在完成开发后我们做了第二次一致性检验。

在一致性检验的过程中事实上发现了不一致的地方，我们分析原因主要是在开发过程中遇到实际实现难度比较大的设计方法时采取其它不符合设计模型的方法实现所导致的，并在迭代的过程中通过为设计模型的微调保证一致性。

#### 4.3.1 代码与图之间的一致性以及各图之间的一致性

检查项目	检查结果
类图与代码之间的一致性	一致
其余图与代码之间的一致性	GUI 状态图与代码实现有出入，缺少状态转移，部分状态转移的事件不确切
各图之间是否一致	一致

表 4.1 代码开发与设计之间的一致性检验

#### 4.3.2 设计与需求的一致性

需求	检查结果
自动创建默认的 RUCM 规则	一致
规则的增删改查、保存、导入	一致
规则说明书	一致
提取输入的规则 JSON 文件中的字段	一致
提供一系列规则管理的逻辑操作	一致
规则检查	一致
报告生成	一致

表 4.2 设计与需求的一致性检验结果

## 5 项目总结

RUCM 是一种结构化和模板化的需求规格，引入了流程、结构化句型和流程控制机制。本项目以 RUCM 编辑器产生的 rucm 文件作为输入，依据课堂所讲授的 RUCM 规范指定相应的规则，并按照规则来自动检查一个具体的需求违反了哪些规则，同时能够支持规则的设置。并在第二版的迭代过程中，根据老师的指导加入了中文支持与用户规则编辑、图形化前端界面等功能，形成了完整可靠的初代产品。

本项目大致上可以分为四个模块，加载.rucm 文件与规则文件模块，自然语言处理模块，检查模块，规则模板设计模块。加载模块基本完成了加载文件的功能，但是尚存在许多不确定因素没有加以测试，在目前的测试样例中，能够完成正确加载文件的功能。自然语言处理模块由于其自身具有不确定性，无法保证完成了解析句子成分的功能，但是在现有测试中，能够对绝大多数输入实现正确的划分，基本实现了功能。检查模块实现了根据规则.rucm 文件的功能，在当前测试中表现良好。规则设计模板模块的功能是给用户提供详尽的规则制定模板以及相应的说明书，说明书尚不完善，模板已经提供，还需完善。总体上来说，各个模块都实现了基本的功能，项目基本实现了需求分析阶段中提出的功能性需求，实现与设计阶段的一致性基本满足。在实现过程中，由于各个模块的接口不一致、类的约束不明确等一系列原因，出现了各个模块之间变量命名不一致、功能无法实现等问题。通过返回设计阶段，完善设计，解决了这些问题。

## 附录 A rule-template(规则模板) 说明

- rule 文件有一个 dict 组成，其中有两个字段，分别是 default 和 user-def。分别代表默认规则列表和用户定义规则列表。
- default 字段下的数据尽量不要改，唯一可以改的是 status 字段，当 status 为 false 是，规则处于禁用状态
- 用户规则列表由用户规则聚合而成，一个用户规则下的字段包括：
  - id: 用户规则 id，不能重复
  - applyScope: 规则作用域,取值包括“actionStep”(flow 中的句子),“allSentence”(RUCM 中所有句子)
  - simpleRules: 一个用户规则可以被看做多个简单规则的聚合，简单规则字典信息包括：
    - subject: 规则要检查的对象，他的取值包括:
    - subject\_Val: 主语的值
    - subject\_count: 主语的数量
    - object\_Val: 宾语的值
    - object\_count: 宾语的数量
    - verb\_count : 动词的数量
    - verb\_tense: 动词的时态
    - strs: 句子中所有的词
    - modal\_verb\_count: 情态动词的数量
    - adverb\_count: 副词的数量
    - pronoun\_count: 代词的数量 operation: 对检查对象的约束，可以是 in/not-in/le/lt/eq/neq/ge/gt, 分别表示约束对象在/不在/小于等于/小于/等于/不等于/大于等于/小于
    - val: 约束列表

例如 {"subject":"subject\_Val","operation":"in","val":["system"]} 表示主语中不能出现 system 字眼

列表中除了出现正常字符串以外，还可以出现 “\$actor” 代表 RUCM 中的 actor

- 注意: 当 subject 字段为 XX\_count 的时候，val 字段中只能出现数字，当 operation 为除了 in/notin 以外的字段时候，val 字段中只能出现数字。

- status: 当 status 为 false 时, 规则处于禁用状态
- operation: 综合简单规则的逻辑操作
  - 第一个字符代表对整个检查结果的操作 “-” 代表无操作, “!” 代表取非操作
  - 第一个字符往后分别代表对于简单规则之间的逻辑操作, “&” 代表与操作, “|” 代表或操作
- description: 规则描述

## 附录 B 领域分析报告

# RUCM 领域分析报告

高等软件工程 2018

敏敏特穆尔队

## 1. 项目要求阐述

### 1.1 项目输入

以 RUCM 编辑器产生的 JSON 文件。该 JSON 文件结构如图 1 所示，其中包括用例（Use Case）信息、参与者（Actor）信息、关系（Relationship）信息等。其中，用例包括用例模板中的各说明字段（见术语 3.3 RUCM）。

另外，系统提供规则编辑说明书，用户可根据说明书填写规则的 JSON 文件。

### 1.2 项目目的

设计 RUCM 规则检查工具。

### 1.3 项目功能要求

- **维护、管理规则库**

规则库包括 26 条默认 RUCM 规则（见术语 3.3 RUCM）和用户自定义规则。对规则库的维护需要项目能够在系统开始时自动创建默认的 RUCM 规则。在规则的维护方面，要求项目能够建立具有特定格式的规则结构。该规则结构包

括规则的类别（如自然语言规则、关键词规则）、对应的关键字段主体以及逻辑操作等内容。值得注意的是，默认规则以及可能出现的用户自定义规则中各规则的作用有较大差别。通过规则的检查对象以及规则的具体作用，可将规则进行不同的分类。这要求系统在设计时需要详细定义各规则的类别，对各类别分别制定不同的检查方法。

在规则编辑方面，要求项目能够：(1) 在规则说明书中为用户的规则管理提供关键字词（规则主语）、作用范围、规则类别以及逻辑操作类别，供用户编写规则 JSON (2) 提取输入的规则 JSON 文件中的相关字段，建立对应数据结构。

(3) 提供一系列规则管理的逻辑操作（如必须是、可以是、不能是、包含等逻辑关系）。上述要求旨在使用户能够通过 RUCM 规则中的模板项、关键字词等信息进行规则上的编辑，同时结构化规则信息，方便系统规则库的维护与管理。

## ● 规则检查

该项要求需要系统按照规则自动检查一个具体需求是否出现违反规则情况，标明所违反规则的类别并定位。一个 RUCM 模型从结构上包括 RUCM 模板上的各字段内容及规则（见术语 3.3 RUCM）。但是，输入的以 JSON 格式描述的 RUCM 模型只包括用例模型以及各用例的模板信息，而没有规则信息。本系统的主要功能是通过预设的规则（默认规则或用户添加规则）对输入进行规则检测。

必须注意的是，因为系统中各规则的类别不同，其处理方法也不尽相同，这要求系统能够根据不同的规则类别，使用不同的检测方法，对每一用例，逐条使用规则进行检查。默认的 26 条 RUCM 规则按对象及作用可划分为自然语言规则以及关键字规则，分别用来约束自然语言和触发关键字。其中，自然语言规则

可进一步分为对于事件流的规则要求以及对于用例的全部组成的要求。对于自然语言规则，要求系统能够正确调用自然语言处理工具，按照作用范围分别对用例的事件流字段或用例中所有字段进行分词处理、词性检查以及语法分析，检查字段是否符合自然语言规则。对于关键字规则，要求系统能够准确识别控制结构内所要求的关键字信息。对于用户指定的规则，需要系统在为用户提供关键字选择以及逻辑操作前，设计好可能产生的规则形式模板以及规则语义分析方法。在用户指定或更改规则时，根据已制定的语义信息对规则范围内的用例信息进行检查。

每当在某用例中检测出规则违反行为，则需要标记违反规则情况，记录违反的具体规则，并标记该用例的名称及引用。在系统检查完毕后，需要将所有记录进行保存并输出到报告文件。

## ● 报告生成

在规则检查结束后，需要系统将中的违反规则情况的记录进行整合并输出到文件。输出内容包括：

- (1) RUCM 模型规则检查结果。检查结果包括“RUCM 模型符合规则”、“RUCM 模型未通过规则检查”、“规则检查失败”三种情况，分别表示输入的 RUCM 模型中没有不符合规则的情况、输入的 RUCM 模型中存在不符合规则的情况以及规则检查失败的情况。其中，规则检查失败的情况有输入 JSON 不符合格式要求、规则检查程序运行错误等情况。
- (2) RUCM 模型中不符合规则的情况。该部分是结果“RUCM 模型未通过规则检查”的拓展，包括内容有不符合规则的记录数量及记录信息。每条记录信息包括记录编号、记录所在的用例名称（含引用）、记录所在的用例模板

字段名称（含引用）以及违反的规则内容。

## 1.5 项目性能要求

综合考虑工程大小、实际应用环境以及当前额主流机器软硬件配置情况，性能要求可忽略。

## 1.6 项目输出

输出规则检查的结果和关于违反规则的报告。

# 2. 领域定位

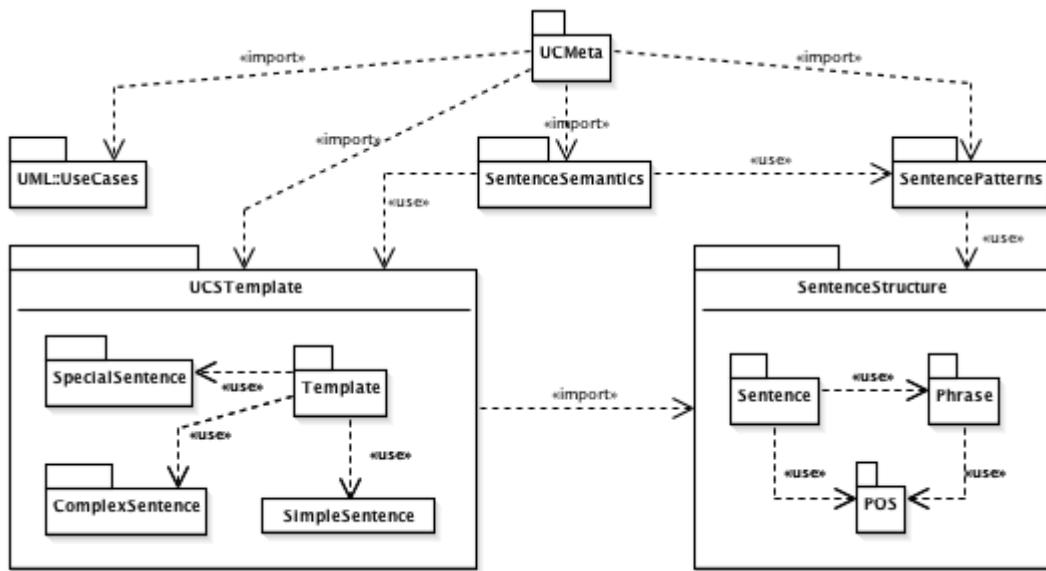
RUCM (Restricted Use Case Modeling)需求建模方法是一种基于用例的方法，由结构化的用例规约模板和一组限制规则构成。于 2013 年由 YUET, BRIAND L C, LABICHE Y 等人提出结构化的模板定义了在用例规约中需要描述的内容，如表 1 所列。限制规则包括自然语言规则与关键字。实验证明，RUCM 方法可以有效地降低需求规约的二义性，同时又保留自然语言易理解和易使用的优点。

表 1 RUCM 用例规约模板

Use Case Name	用例名称,通常以动词开头	
Brief Description	用例内容的简要描述	
Precondition	用例的前置条件,即用例执行前必须满足的条件	
Primary Actor	主要参与者	
Secondary Actors	次要参与者	
Dependency	依赖关系,描述与其他用例之间的包含、扩展关系	
Generalization	泛化关系,描述与其他用例之间的泛化关系	
Basic Flow	Steps	事件流步骤
	PostCondition	基本流的后置条件
	GuardField	引用流中的步骤号或守护条件
Alternative Flows	Steps	事件流步骤
	PostCondition	分支流的后置条件

北京航空航天大学已经针对 RUCM 方法开发出相应的 建模工具(下载地址: <http://zen-tools.com/rucm/Editors.html>), 该工具支持 RUCM 用例模板和约束规则, 并能够对 RUCM 关键词及其用法进行检查。RUCM 建模工具能够 将 RUCM 需 求 转 化 为 UCMeta 的 实 例 模 型 , 从 而 支 持 对 RUCM 需求模型的自动化分析。

UCMeta 定义了 RUCM 用例模型的所有必要概念及其关 系 ,由用例图、用例规约模板、自然语言元模型组成。用例规约模板元模型元素如下图所示。用例规约中一个重要的概念是事件流 (FlowOfEvent), 它按时间次序描述了参与者和系统交互的具体步骤, 由语句集合 (steps)构成 。



一般而言，需求完整 意味着:1)需求包括了软件应该实现的所有功能 ;2)需求为所有有效的输入定义了响应 ;3)需求描述涉及的图表都有完整标识和描述;4)需求描述中涉及的计量术语和单位有完整定义。由于不同软件的需求描述内容采用的描述方式存在差异，因此在需求确认活动中对完整性的确认有所差异。需求确认确保需求描述了系统利益相关者对系统的所有功能要求;需求验证依据现有需求，结合需求描述方法，从需求本身逻辑中找到缺失的内容。例如，在使用用例的需求描述中，要求每个用例和动作都有前置条件和后置条件;在涉及资源描述的用例需求中，要求每个资源都应 该在用例的场景描述中出现。

需求一致性指需求描述内部是否有逻辑矛盾的特性，如果有逻辑矛盾，则意味着需求不一致，否则为需求一致，任 何需求建模方法都要有自己的一套规则来检查需求模型的一致性。需求不一致问题主要表现为：对外部对象（如 资 源或者外部实体)的要求或约束不一致;需求描述动作之间在 描述逻辑或执行时序上不一致;对一个对象的描述和术语使 用不一致。

用例规约一般采用自然语言描述，自然语言会带来二义性问题，因此不同的需求描述方法都设计了相应的约束来 规范自然语言的使用以减少产生二义性的机会。主要约束有 3 类 :1 ) 通过自然语言 使用 规 则 和 关 键 字 来 限 制 ， 这 些 方法主要定义需求中频繁使用的关键字、词组和句子结构;2) 通过描述指导规则来限制，这些方法总结了书写自然语言 需求规约时的指导规则，比如主要子句的动词应该用主动语 态 ;3)针 对 领 域 的 语 言 使 用 模 式 。 所 谓 不 符 合 需 求 描 述 方 法 规 范， 是 指 没 有 按 照 描 述 方 法 进 行 描 述，可 能 会 使 得 需 求 不 能 很 好 地 被 理 解， 分 析 与 验 证 时 会 出 现 错 误。

### 3. 主要术语及解析

#### 3.1 JSON

JSON (JavaScript Object Notation) 是一种由道格拉斯·克罗克福特 (Douglas Crockford)构想和设计、轻量级的数据交换语言，该语言以易于让人阅读的文字为基础，用来传输由属性值或者序列性的值组成的数据对象。尽管 JSON 是 JavaScript 的一个子集，但 JSON 是独立于语言的文本格式，并且采用了类似于 C 语言家族的一些习惯。JSON 数据格式与语言无关，脱胎于 JavaScript，但目前很多编程语言都支持 JSON 格式数据的生成和解析。JSON 的官方 MIME 类型是 application/json，文件扩展名是 .json。JSON 格式是 1999 年《JavaScript Programming Language, Standard ECMA-262 3rd Edition》的子集合，所以可以在 JavaScript 以 eval()函数 (javascript 通过 eval () 调用

解析器) 读入。不过这并不代表 JSON 无法使用于其他语言，事实上几乎所有与网页开发相关的语言都有 JSON 函数库。

JSON 用于描述资料结构，有两种结构存在：

- 对象 (object)：一个对象包含一系列非排序的名称 / 值对(pair)，一个对象以{开始，并以}结束。每个名称 / 值对之间使用:分割。
- 数组 (array)：一个数组是一个值(value)的集合，一个数组以[开始，并以]结束。数组成员之间使用,分割。

具体的格式如下：

- 名称 / 值 (pair)：名称和值之间使用: 隔开，一般的形式是：

{name:value}

一个名称是一个字符串；一个值(value)可以是一个字符串(string)，一个数值(number)，一个对象(object)，一个布尔值(bool)，一个有序列表(array)，或者一个 null 值。

- 字符串：以""括起来的一串字符。
- 数值：一系列 0-9 的数字组合，可以为负数或者小数。还可以用 e 或者 E 表示为指数形式。
- 布尔值：表示为 true 或者 false。
- 值的有序列表 (array)：一个或者多个值用,分割后，使用[，]括起来就形成了这样的列表，形如：[value, value]

JSON 的格式描述可以参考 RFC 4627。

### 3.2 用例模型 (UCM, Use Case Modelling)

用例模型以用例的角度描述一个系统的功能需求，是对系统的目标功能（用例）和环境（参与者）的建模。用例模型本身是以系统功能为目标，从外部来观察其实现或者操作过程。用例分析方法是一种和用户交流并获取需求的技术和手段，具有简单直观、容易上手的特点。与结构化分析方法(SA)中的数据流+数据字典方法、传统的面向对象软件工程(OOSE)方法中的对象模型方法相比较，用例模型更容易被用户和开发者理解，从而更便于对系统的需求取得共识，更适合作为系统分析人员和用户之间交流的工具。用例模型主要由参与者（Actor）、用例（Use Case）及其关系构成，以下为其解释。

- 用例（Use Case）：用例是从使用者的角度或者说从系统外部观察系统的能力。它是系统功能抽象的使用案例，描述了系统功能的使用过程或者与用户的交互过程。用例定义了一组使用案例，每个案例都是系统产生的一组动作，这组动作对参与者产生可观察的数值结果。
- 参与者（Actor）：参与者代表了所有与系统交互的角色。参与者可以代表系统用户，也可以代表系统之外其他的任何事物，如其他软件系统、打印机等。参与者可以是系统之外但和系统有关的任何元素，包括影响系统和受系统影响的任何元素。利用参与者可以帮助定义系统边界(定界或者界定系统)。凡是抽象成参与者的都是系统之外的元素，凡是抽象成用例的都是系统之内的功能。

### **3.3 Use Case Specifications(UCSs)**

UCSs 通常是一段文字文档，遵从一个特定的用例模板，用于帮助人们理解和检查用例。

### **3.4 RUCM (Restricted Use Case Modeling)**

RUCM 是一种用例建模方法，其目的在于限制用户编辑 UCSs，达到减少二义性，增强用例模型的可理解性的目的，促进对模型的自动化分析。RUCM 由结构化的用例模板和一组限制规则组成。用例模板整合了相关工作的内容，包含了常规模板的常用字段，而且能更好地明确用例的事件流结构。

一个标准的 RUCM 表的字段包括 use case name (用例名称)、brief description(用例简介)、precondition (用例的使用条件)、primary/secondary actor (主要/次要使用者)、dependency (依赖)、generalization (泛化)、basic flow (基本流)、Alternative flow(备选流程)下面是针对主要字段以及相关术语的简介：

- use case name

该字段给出了用例的名称。它通常以动词（例如，Withdraw Fund）开头，并且应与用例图中相同用例的名称一致。

- Brief Description

该字段在一个简短的段落中总结了用例。

- Precondition

用例的前提条件指定在用例开始之前必须始终为真的内容。

- Primary Actor

用例的主要角色通常是发起对软件使用的人/系统

- Dependency

这个字段说明了<<include>>和<<extend>>这两个用例之间的关系

- Generalization

这个字段说明了用例间的泛化关系

- Basic Flow

用例的基本流程描述了满足需求者要求的主要动作序列。它通常不包括任何条件或分支。建议单独描述替代流程中的条件和分支。基本流程由一系列步骤和后置条件组成。每个 UCS 只能有一个基本流程。

操作步骤可以是以下五种交互之一：

- 1) 主要参与者->系统：主要参与者向系统发送请求和数据；
- 2) 系统->系统：系统验证请求和数据；
- 3) 系统->系统：系统改变其内部状态（例如，记录或修改某些内容）；
- 4) 系统->主要参与者：系统回复主要参与者的请求；

所有步骤按顺序编号。这意味着每个步骤在下一个步骤开始之前完成。如果需要表达条件，迭代或并发，则应应用指定为限制规则的特定关键字。

- Alternative flows

备选流程主要描述基本流程之外的分支。

表 1 为一结构化的用例模板。限制规则包括对自然语言的限制和被特定结构所强制要求的关键字的限制等，其中，表 2、表 3 表示对自然语言的限制规则，表 4（除第 26 条）表示对控制结构所要求的关键字。

Table 1 Use case template

<b>Use Case Name</b>	The name of the use case. It usually starts with a verb.
<b>Brief Description</b>	Summarizes the use case in a short paragraph.
<b>Precondition</b>	What should be true before the use case is executed.
<b>Primary Actor</b>	The actor which initiates the use case.
<b>Secondary Actors</b>	Other actors the system relies on to accomplish the services of the use case.
<b>Dependency</b>	Include and extend relationships to other use cases.
<b>Generalization</b>	Generalization relationships to other use cases.
<b>Basic Flow</b>	Specifies the main successful path, also called “happy path”.
<b>Steps (numbered)</b>	Flow of events.
<b>Postcondition</b>	What should be true after the basic flow executes.
<b>Specific Alternative Flows</b>	Applies to one specific step of the basic flow.
<b>RFS</b>	A reference flow step number where flow branches from.
<b>Steps (numbered)</b>	Flow of events.
<b>Postcondition</b>	What should be true after the alternative flow executes.
<b>Global Alternative Flows</b>	Applies to all the steps of the basic flow.
<b>Steps (numbered)</b>	Flow of events.
<b>Postcondition</b>	What should be true after the alternative flow executes.
<b>Bounded Alternative Flows</b>	Applies to more than one step of the basic flow, but not all of them.
<b>RFS</b>	A list of reference flow steps where flow branches from.
<b>Steps (numbered)</b>	Flow of events.
<b>Postcondition</b>	What should be true after the alternative flow executes.

Table 2 Restrictions (R1-R7)

#	Description	Explanation
R1	The subject of a sentence in basic and alternative flows should be the system or an actor.	Enforce describing flows of events correctly. These rules conform to our use case template (the five interactions).
R2	Describe the flow of events sequentially.	
R3	Actor-to-actor interactions are not allowed.	
R4	Describe one action per sentence. (Avoid compound predicates.)	Otherwise it is hard to decide the sequence of multiple actions in a sentence.
R5	Use present tense only.	Enforce describing what the system does, rather than what it will do or what it has done.
R6	Use active voice rather than passive voice.	Enforce explicitly showing the subject and/or object(s) of a sentence.
R7	Clearly describe the interaction between the system and actors without omitting its sender and receiver.	

Table 3 Restrictions (R8-R16)

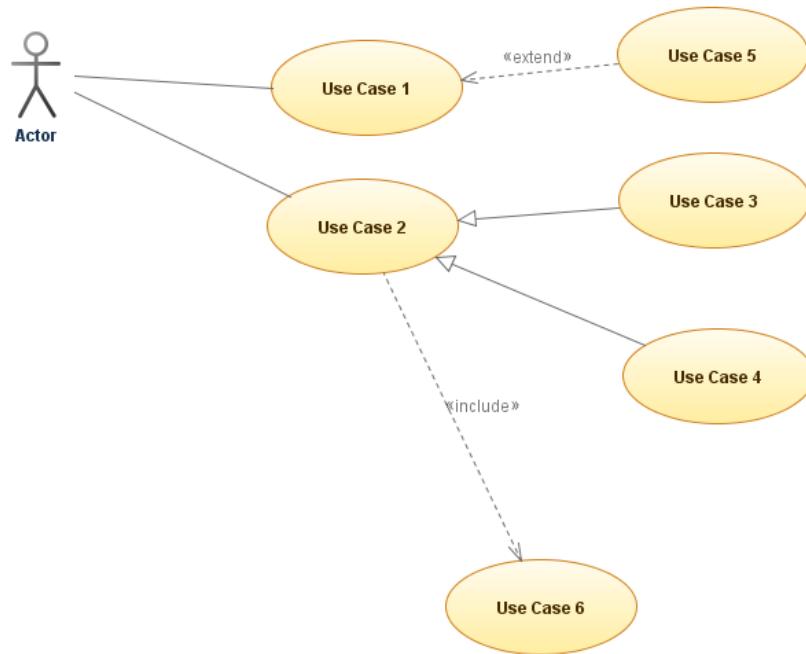
#	Description	Explanation
R8	Use declarative sentence only. “Is the system idle?” is a non-declarative sentence.	Commonly required for writing UCs.
R9	Use words in a consistent way.	Keep one term to describe one thing.
R10	Don’t use modal verbs (e.g., <i>might</i> )	Modal verbs and adverbs usually indicate uncertainty; therefore metrics should be used if possible.
R11	Avoid adverbs (e.g., <i>very</i> ).	
R12	Use simple sentences only. A simple sentence must contain only one subject and one predicate.	Facilitate automated NL parsing and reduce ambiguity.
R13	Don’t use negative adverb and adjective (e.g., <i>hardly</i> , <i>never</i> ), but it is allowed to use <i>not</i> or <i>no</i> .	
R14	Don’t use pronouns (e.g., <i>he</i> , <i>this</i> )	
R15	Don’t use participle phrases as adverbial modifier. For example, the italic-font part of the sentence “ATM is idle, <i>displaying a Welcome message</i> ”, is a participle phrase.	
R16	Use “the system” to refer to the system under design consistently.	Keep one term to describe the system; therefore reduce ambiguity.

Table 4 Restrictions (R17-R26)

#	Description	#	Description
R17	INCLUDE USE CASE	R22	VALIDATE THAT
R18	EXTENDED BY USE CASE	R23	DO-UNTIL
R19	RFS	R24	ABORT
R20	IF-THEN-ELSE-ELSEIF-ENDIF	R25	RESUME STEP
R21	MEANWHILE	R26	Each basic flow and alternative flow should have their own postconditions.

### 3.5 RUCM 模型的 JSON 文件表示

RUCM 模型包括结构化的用例模板以及一组用例规约。现有的 Eclipse 插件 RUCM 中，RUCM 模型的表示结构为 JSON 格式。以下用例图为例。



其中，Use Case 5 为已完成用例模板填写的用例，其填写内容如下。

Use Case Specification									
Use Case Name	Use Case 5								
Brief Description	Helloooooooooo Wooooooooorld!!								
Precondition	None								
Primary Actor	Actor								
Secondary Actors	None								
Dependency	EXTENDED BY USE CASE Use Case 5								
Generalization	None								
Basic Flow	Steps								
"Flow 1" ▼	<table border="1"> <tr> <td>1</td><td>Do sth.</td></tr> <tr> <td>2</td><td>System VALIDATES THAT it is a good day.</td></tr> <tr> <td colspan="2">Postcondition</td></tr> <tr> <td colspan="2">Success</td></tr> </table>	1	Do sth.	2	System VALIDATES THAT it is a good day.	Postcondition		Success	
1	Do sth.								
2	System VALIDATES THAT it is a good day.								
Postcondition									
Success									

最后生成的 JSON 文件结构分别对模型中的元素、模型描述、模型名称、模型版本、引用关系、模型图组成进行了表示。表示内容如下。

表示内容	表示结构
模型元素——Actor	<ul style="list-style-type: none"> <li>- content</li> <li>  - description //Actor 描述</li> <li>  - name //Actor 名称</li> <li>- type:"Actor"</li> </ul>
模型元素——Use Case	<ul style="list-style-type: none"> <li>- content</li> <li>  - description //Use Case 描述</li> <li>  - extend/include //Use Case 的 extend/include 关系</li> <li>  - name //Use Case 名称</li> <li>  - specification //模板内容</li> </ul> <p>-type:"Use Case"</p>
模型元素——连线	<ul style="list-style-type: none"> <li>- content</li> <li>  - use case/actor //连线两端的对</li> </ul>

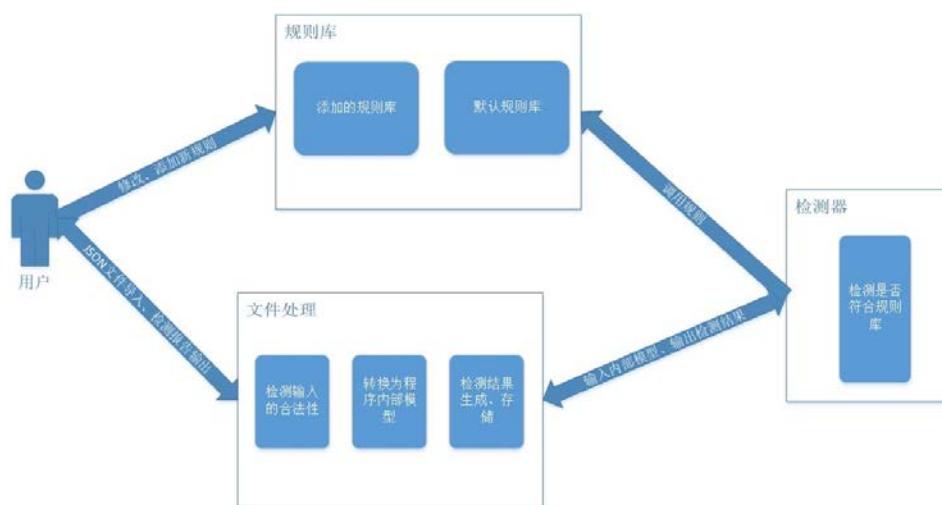
	<p>象</p> <ul style="list-style-type: none"> <li>- description //连线描述</li> <li>- name //连线名称</li> </ul> <p>-type:"Relationship/Generalization"</p>
模型描述	<ul style="list-style-type: none"> <li>- description</li> <li>- content //对模型的描述</li> <li>- type //content 的类型 (一般为 String)</li> </ul>
模型名称	<ul style="list-style-type: none"> <li>- name</li> <li>- content //模型名称</li> <li>- type //content 的类型 (一般为 String)</li> </ul>
模型图	<ul style="list-style-type: none"> <li>-diagrams</li> <li>- nodes //图中的节点 (用例、参与者)</li> <li>- links //图中的连线</li> <li>- ucModel //所属模型</li> <li>- top/bottom/width/height //位置信息</li> </ul>

## 引用关系

```
"reference": [
    {"path": "<root>.modelElements[2]"},
    {"path": "<root>.modelElements[10]"},
    {"path": "<root>.modelElements[0]"},
    {"path": "<root>.modelElements[1]"},
    {"path": "<root>.modelElements[5]"},
    {"path": "<root>.modelElements[6]"},
    {"path": "<root>.modelElements[9]"},
    {"path": "<root>.modelElements[3]"},
    {"path": "<root>.diagrams[0].nodes[0]"},
    {"path": "<root>.diagrams[0].nodes[2]"},
    {"path": "<root>.modelElements[4]"},
    {"path": "<root>.diagrams[0].nodes[1]"},
    {"path": "<root>.diagrams[0].nodes[3]"},
    {"path": "<root>.modelElements[7]"},
    {"path": "<root>.diagrams[0].nodes[4]"},
    {"path": "<root>.modelElements[8]"},
    {"path": "<root>.diagrams[0].nodes[5]"},
    {"path": "<root>.modelElements[9].extend[0]"},
    {"path": "<root>.diagrams[0].nodes[6]"},
    {"path": "<root>.modelElements[2].include[0]"},
    {"path": "<root>"}
]
```

模型中各组成的保存路径

## 4. 领域系统架构分析



该系统涉及四个方面，如图所示。

用户为软件的使用者，包括 RUCM 建模人员、RUCM 规则制定人员等。

规则库模块对 RUCM 的约束规则进行管理，包括读取规则、写入规则。其中包括两部分，添加的规则库和默认规则库。添加的规则库为用户添加的规则集合，默认规则库为 26 条对自然语言、控制结构的限制规则。

文件处理模块用于管理文件输入输出，主要分为三个模块。文件输入为 JSON 文件，需要检测其合法性，之后转化为程序内部模型，也即特定的数据结构，以便于接下来检测其是否满足约束规则。另一方面，文件处理模块负责管理文件的输出，文件输出为检测结果报告，用户可以选择输出模式。

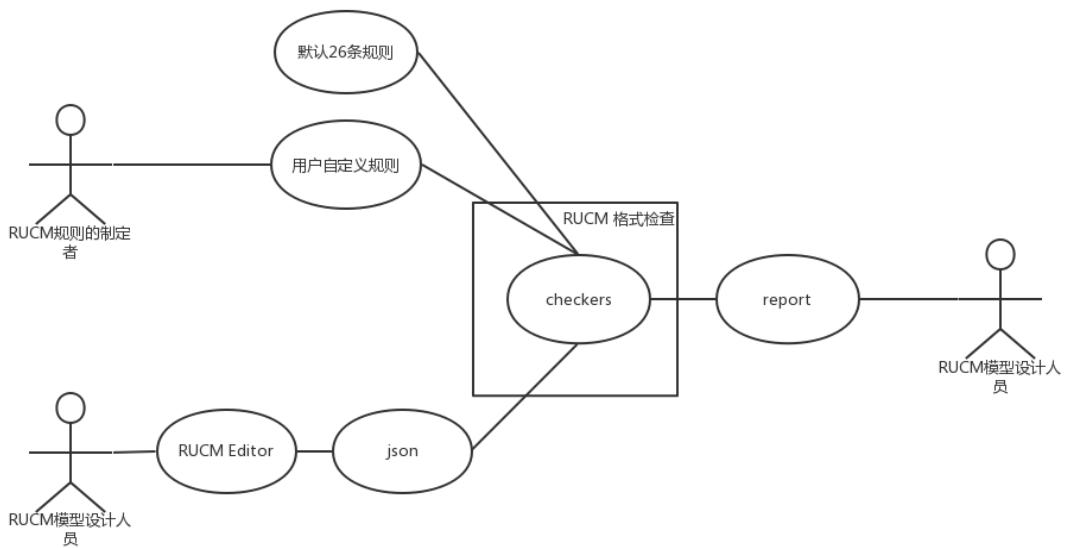
检测器模块为程序的核心实现。检测器结合规则库对转换后的程序内部模型进行检测，包括对 26 条默认规则的检测和对用户新添加规则的检测。并且生成检测报告，提交给文件处理模块，最终输出给用户。

## 5. 领域系统运行环境

系统所处的运行环境主要为 RUCM 建模环境。环境中的参与者包括 RUCM 约束规则制定与研究人员、RUCM 建模人员、RUCM 模型数据分析人员、RUCM 模型以及检查系统。在该环境中，RUCM 规则设计与研究人员按照实际需求设定约束规则 Rules(R)。RUCM 建模人员根据这些约束规则 R 的约束使用 RUCM 建模软件建立用例模型 Model(M)，模型输出为结构化的 JSON。RUCM 能够有效减少需求规约的二义性，同时能够保持自然语言易理解和易使用的优点。在建模后，需要利用其中的限制规则 R 对建模结果 M 进行检查。系统对建模后产生的 JSON 进行检查，并指出其中不满足 RUCM 规则约束条件 R 的地方，供 RUCM 建模人员进行修改，同时给项目业主（甲方）提供约束规则 R 的审查规则，方便

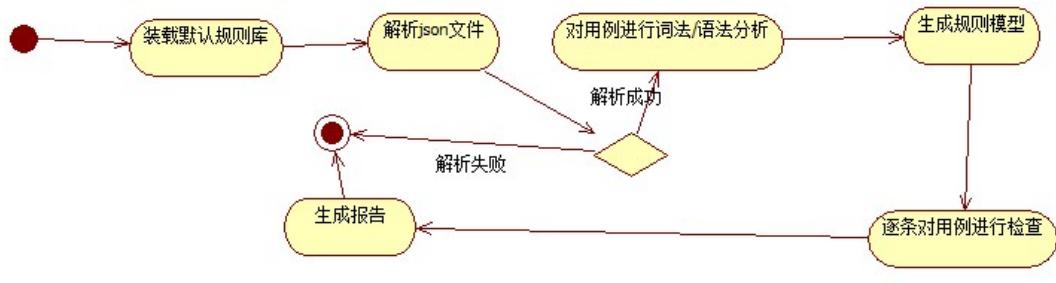
后续使用。系统主要接收、处理的数据为由 RUCM 建模软件产生的 JSON 数据和系统本身需要管理维护的规则库 R。该 JSON 数据主要是 RUCM 模型中的用例信息。每一用例信息由用例规约模板中的各字段组成，包括用例名、用例简介、前置条件、主要参与者、次要参与者、与其他用例的依赖、泛化关系、事件流等。同时，系统能够接收规则制定者根据需求对约束规则的添加、修改、删除。系统产生的输出为规则检查报告，该报告主要用于对 RUCM 建模人员所提交的 JSON 数据中违反限制规则的情况进行反馈，包括违反的限制规则编号、违反限制规则的用例名称等。另外，系统为需求建模人员提供可视化操作界面，用来展示现有规则、提供规则编辑方法及反馈处理报告。

RUCM 模型的 26 条默认约束规则是 RUCM 模型设计工具在设计之初通过专家们的分析研究确定的，但是随着技术的发展与实际需求的变化，这些规则中有部分规则可能不再适用，还可能在实际的使用过程中有了新的普遍的约束规则需求，面对这些问题，研究人员都可以借助本系统进行各种相关的规则约束实验研究。本系统在使用的过程中必然积累大量的私有约束规则数据库，在用户知情并同意的情况下本系统还可以收集整理这些私有规则提供给研究人员，用于研究各行各业具体的约束规则需求与建模设计需求，从而更好地改进 RUCM 建模工具，提高其功能与使用性能。

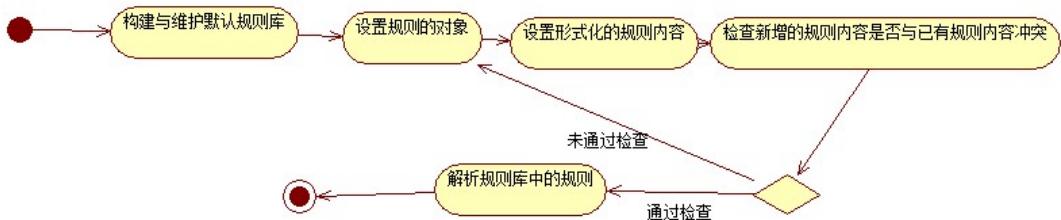


该系统以 checkers 为核心，checkers 以规则和 RUCM 模型的 json 文件为输入。其中，规则根据来源可以分为 2 种，用户自定义规则和默认 26 条规则。用户自定义规则由用户产生。这里的用户主要指的是 RUCM 规则的制定者。默认 26 条规则在第 3 部分有详细介绍，另外所有的 26 条规则和更详细的介绍和示例都可以在 RUCM manual 中找到。RCUM 默认设计人员使用 RUCM Editor 绘制编辑 RUCM 模型，之后可以用 Editor 的导出功能以 json 格式导出 RUCM 模型。该 json 文件作为 checkers 的另一输入，也就是该系统需要检查格式的目标文件。checkers 的输出为一个 report。report 是检查的结果。对于不符合默认规则或自定义规则的 json 输入，report 会给出 json 输入的哪一行违反了哪条规则，以方便 RUCM 模型设计人员修改更正自己的 RUCM 模型。对于符合所有规则的 json 输入，report 会反馈正确信息。

## 6. 领域系统主要流程分析



主要流程概览



规则库管理活动图

### 6.1 规则库维护与管理

规则库的维护与管理包括默认库的构建与维护以及自定义规则的增加、修改、删除、查看操作。同时，规则库在规则发生更变时将自动检测规则间的关联约束（包括若干规则间发生冲突、若干规则间产生重复以及规则间的包含、连接关系），并向用户展示关联约束信息，对于冲突、重复的信息，将提示用户修改。规则库维护与管理流程如下。

#### 1) 构建与维护默认规则库

在 RUCM 中，默认规则库共有 26 条规则，按作用可分为两类：用于自然语言约束的规则、用于控制流程关键字的规则。其中，用于自然语言约束的规则

按照作用范围可分为针对事件流描述的规则以及针对用例全部字段描述的规则。

考虑到默认规则在语义上难以被计算机解释，另一方面，考虑到默认规则的常用性，系统将默认规则库内置于代码，随系统的启动自动加载。在构建默认规则时，系统会生成不同的分类号，区分这些规则的用途，以便后续调用不同的规则处理方法进行规则检查。因此，默认的规则库包括如下条目：规则 id，规则的作用范围，规则内容，规则的分类号以及是否启用该规则。规则 id 表示了规则的序号，规则作用范围指定了规则检查时所处理的条目内容，规则的分类号区分该规则是自然语言处理类规则或关键字规则。在维护方面，用户可以决定是否启用某条规则，当用户拒绝启用一条规则时，该规则的“是否启用该规则”条目置为 False。

## 2) 接收用户自定义规则，加入新规则库

为了减小用户自定义规则中由自然语言带来的不确定性，系统将用户自定义规则的结构进行了一定的约束。为了表示这些规则，我们提出了规则模型的概念。一条规则模型包含的基本属性包括：

i) 规则的作用域。规则的作用域表示规则的具体作用范围。RUCM 模板中各字段有不同的规则要求，因此，每条规则适用的字段不同。比如，默认的 26 条规则中，R1-R7 字段只能作用于各类事件流的 Steps 字段；而 R8-R16 能够作用于整个用例模板的所有句子中，即 R1-R7 的作用域为 Steps 字段，而 R8-R16 的作用域为整个用例模板。

ii) 规则的格式化结构。对于一条规则，我们定义了一个简单的结构，该结构包含了规则的约束对象、逻辑约束操作和逻辑约束范围。如“用例名（字段）不

包括‘打开’”，该规则的约束对象为用例名（字段），逻辑约束操作为“不包括”，逻辑约束范围为“打开”。其中，约束对象可以是某一个字段，也可以是某字段中的句子，同时也可是某字段中的一个关键字词。系统将为用户提供特定的逻辑约束操作，以约束规则的制定与修改范围。

## 6.2 需求 JSON 文件解析及模型生成

输入的 JSON 文件为一文字流，无法对其直接操作，因此，需要根据 JSON 文件的内容进行再次建模，将其文字内容统一到一特定的数据结构中，以便进行下一步的规则检查。该流程包括以下内容：1) 读取用户输入 JSON 文件。2) 检查 JSON 文件正确性与解析 JSON 文件结构。调用相关 JSON 解析工具，对 JSON 文件进行解析。如果解析失败，则根据 JSON 解析工具提供的错误报告判断其正确性是否出现问题。3) 根据用例模板结构，整理用例的表头结构。4) 将 JSON 内容中的每个用例变为结构化的数据结构。

## 6.3 规则检查

对于构建 RUCM 模型中各用例，逐一检查其是否满足要求。其中，按术语 3.3 RUCM 中规则表格，可对规则检查方法做如下划分。

- 1) 默认规则库中的自然语言规范检查。检查规则对应表格中 R1-R16 项。
  - i. 针对每个实例的自然语言部分进行分词及词法分析。针对新要求内容的每一个词语进行归类。将词语分为动词，名词，形容词，关键字，未知类别等类别。并且每个类别中可以有子类别，如动词对动词进一步分析归类，归类到现在进行时，过去时等时态。

- ii. 对模型的名词寻找用例模型领域对应，寻找模型涉及的对象以及 actor。
  - iii. 检查用例中是否包含无法归类的词语。
  - iv. 对用例的每一个条目进行语法分析。将条目变为形式化模型。
  - v. 检查形式化模型是否符合 16 条规则中的词法规则。
  - vi. 检查形式化模型是否符合 16 条规则中的语法规则。
  - vii. 检查形式化模型的动作是否符合 16 条规则中的动作规则。
- 2) 关键字检查。检查规则对应表格 R17-R25 项。
- 3) 属性缺失检查。检查规则对应表格 R26 项。
- 4) 用户自定义规则检查。该检查部分按照规则模型中的各字段，从规则的作用域、规则对象、逻辑约束操作以及逻辑约束范围上逐一对用例进行检查。具体地，固定一个规则，提取该规则的作用域，对每一个用例在作用域内的字段进行检查。检查包括判断作用域内是否存在规则对象，若存在，则判断其是否符合逻辑约束，即是否满足逻辑约束操作在逻辑约束范围上的条件；若不存在，则继续对下一用例进行检查。重复该步骤，直到用例检查完毕后，再使用下一条规则进行检查。

## 6.4 生成报告

在规则检查结束后，需要系统将中的违反规则情况的记录进行整合并输出到文件。输出内容包括：

(1) RUCM 模型规则检查结果。检查结果包括“RUCM 模型符合规则”、“RUCM 模型未通过规则检查”、“规则检查失败”三种情况，分别表示输入的 RUCM 模型中没有不符合规则的情况、输入的 RUCM 模型中存在不符合规则的情况以及规则检查失败的情况。其中，规则检查失败的情况有输入 JSON 不符合格式要求、规则检查程序运行错误等情况。

(2) RUCM 模型中不符合规则的情况。该部分是结果“RUCM 模型未通过规则检查”的拓展，包括内容有不符合规则的记录数量及记录信息。每条记录信息包括记录编号、记录所在的用例名称（含引用）、记录所在的用例模板字段名称（含引用）以及违反的规则内容。

## 7. 领域系统用户识别

本系统的主要功能是通过审查分析 RUCM 建模结果 Model 是否满足一个由用户给定的既定规则库 Rules 的来验证 Model 的正确性与有效性，从而确定其是否达成了用户的既定设计目标，并可以在模型中具体地标注是那一部分设计内容不满足何种设计规则约束，因此本系统的主要目标用户是软件工程领域中的建模设计人员以及模型设计过程中的限制规则库维护人员。具体地说，本系统潜在的领域用户有：

### 7.1 RUCM 规则的制定者

RUCM 建模的过程中，总是要保证模型的设计满足一定的规则限制。虽然 RUCM 设计方法中默认的设计了 26 条通用规则，但是针对任何具体的模型设计

过程，都可能会有新的、具体的、针对性的规则需要添加，也可能无需遵守 26 条通用规则中的某些不符合本项目的规则，因此 RUCM 设计过程的甲方（项目需求方业主）可以通过本系统来选择某些默认规则、删除一部分默认规则、增加项目特有的新规则来维护一个私有的规则库，从而可以更好地与乙方（项目设计方）RUCM 设计人员进行沟通，向他们表达本项目的实际需求，进一步可以检验 RUCM 设计人员完成的模型设计结果是否满足了项目的预期需求与规则约束，以简洁高效地完成 RUCM 模型设计任务。

## 7.2 RUCM 模型设计人员

对于 RUCM 的模型设计人员，可以使用本系统检查 RUCM 设计模型是否满足了给定的约束规则库，同时可以充分利用本系统可以在模型中具体地标注是那一部分设计内容不满足何种设计规则约束的功能，快速地筛查设计模型中不满足的设计需求的内容，并在完成修改设计后再次送入系统进行检查，在这个迭代审查分析的过程中多快好省地完成 RUCM 模型设计任务，从而有效提高 RUCM 建模设计的模型质量与设计速度，同时还可以大幅度降低建模设计人员的工作量。

## 7.3 RUCM 建模方法研究人员

对于 RUCM 这样的有效地建模设计工具，除工业界有大量设计开发人员在使用外，学术界亦有许多研究人员在不同层面上研究 RUCM 建模方法。这些研究人员可以通过本系统帮助他们进行大量的研究实验。例如 RUCM 模型的 26 条默认约束规则是 RUCM 模型设计工具在设计之初通过专家们的分析研究确定的，但是随着技术的发展与实际需求的变化，这些规则中有部分规则可能不再适

用，还可能在实际的使用过程中有了新的普遍的约束规则需求，面对这些问题，研究人员都可以借助本系统进行各种相关的规则约束实验研究。

同时本系统在使用的过程中必然积累大量的私有约束规则数据库，在用户知情并同意的情况下本系统还可以收集整理这些私有规则提供给研究人员，用于研究各行各业具体的约束规则需求与建模设计需求，从而更好地改进 RUCM 建模工具，提高其功能与使用性能。

#### **7.4 大规模 RUCM 旧模型挖掘与重构人员**

在一些机构或是企业在长时间从事 RUCM 模型设计行业的过程中必然积累了大量旧有的 RUCM 设计模型。利用本系统可以快速地实现对这些积累的设计模型的验证分析工作，在现有的约束规则下实现对历史数据的快速审查分析与重构，提高设计模型的利用率。同时这通过对大批量现有存量数据的挖掘分析与重构，探索研究在不同的规则约束条件下 RUCM 模型设计中的规律与方法，提供给 RUCM 模型设计人员，从而促进整个行业的发展。

#### **7.5 RUCM 模型的 IDE (集成编辑环境)**

本系统还可以直接嵌入 RUCM 的集成编辑环境 (IDE, Integrated Development Environment) 中，通过事先的配置文件编辑好约束规则库，然后类似于代码自动补全工具与语法高亮、语法分析工具一样，实时的检测与提示设计人员在 RUCM 模型设计过程中出现的违反了某条规则的地方，并提出修改建议，从而帮助设计人员高效快速且高质量的完成 RUCM 模型设计。

## 8. 待开发系统的目 标分析

开发本系统的主要目标是帮助软件开发过程中的建模设计人员进行 RUCM 建模结果的有效性分析，系统以 RUCM 编辑器中产生的描述设计模型 M (Model) 的 JSON 文件为输入，同时在 RUCM 默认 26 条规则的基础上维护、管理一个私有规则库 R (Rules)，进一步实现对该模型 M 的合法性与有效性审查，标记出不符合规则的模型设计部分，并标明具体的违反了何条规则，最后将这些信息整合到检测报告中进行输出。具体的系统设计目标有：

### 8.1 管理、维护规则库

RUCM 建模工具默认包含 26 条约束规则，系统需要时实现在规则库中缺省内置这 26 条规则。但是在实际的建模设计过程中，可能并不需要其中的某些规则，也有可能需要增加新的规则需求，因此本系统需要实现对规则库的管理与维护，需要支持删除某些规则、增加新的规则、修改某些规则，即实现对规则库的增、删、改、查，同时要实现序列化的保存当前规则到磁盘上，也要能够从保存的规则库文件中将其重新导入系统，以实现规则库的可重复利用，提高系统的使用效率。

### 8.2 规则检查

在给定规则库 R 的情况下，面对给定的 RUCM 设计模型 M，系统要实现对模型 M 的审查分析，针对规则库 R 中的全部约束条件逐一检查，确定其是否符合规则库 R 的全部约束条件。

当模型 M 符合规则库 R 的全部约束条件时，返回没有错误的报告。

当在模型 M 中遇到不符合约束规则的设计时，要具体的标识该部分的位置，同时指出违反了规则库 R 中的那一条具体的约束规则。

在这个规则检查的过程中，对于相同的模型 M 在相同的规则库 R 的约束下，应该得到相同的检查分析结果，即要保证检查结果正确性与唯一有效性。

### 8.3 生成报告

在完成规则检查之后，系统还需要将模型 M 中违反规则情况的记录进行整合并输出到文件。

整个报告包含两个部分，第一部分是基于规则库 R 的约束条件下对模型 M 进行检查的总体结果说明，包括“RUCM 模型符合规则”、“RUCM 模型未通过规则检查”、“规则检查失败”三种情况，分别表示输入的 RUCM 模型中没有不符合规则的情况、输入的 RUCM 模型中存在不符合规则的情况以及规则检查失败的情况。其中，规则检查失败的情况有输入 JSON 不符合格式要求、规则检查程序运行错误等情况。

第二部分是 RUCM 模型 M 中不符合规则的具体情况的记录。该部分要实现结果“RUCM 模型未通过规则检查”的拓展，包括内容有不符合规则的记录数量及记录信息。每条记录信息包括记录编号、记录所在的用例名称（含引用）、记录所在的用例模板字段名称（含引用）以及违反的规则内容。

系统要综合以上两部分内容，给出一个完整的检查报告。



附录 C RUCM-Manual



**TAO YUE**

tao@simula.no

Simula Research Laboratory

# Restricted Use Case Modeling Approach

User Manual

April 2010

# Preface

*Use case modeling is commonly applied to document requirements. Restricted Use Case Modeling (RUCM) is a use case modeling approach, which is composed of a set of well-defined restriction rules and a new template. The goal of RUCM is to reduce ambiguity, improve understandability of use case models, and facilitate automated generation of analysis models. The approach has been experimentally evaluated to be applicable, and easier to understand. It yields better models when used by humans.*

*This document gives a description of RUCM, gives an example to illustrate how to apply RUCM, and also provides a quick reference in the end.*

---

---

## TABLE OF CONTENTS

---

---

### INTRODUCTION

### RUCM USE CASE TEMPALTE

### RUCM RESTRICTION RULES

### EXAMPLE OF USE

### QUICK REFERENCE TABLES

---

## Introduction

*A use case model, including a use case diagram and a set of use case specifications, is commonly applied in practice to structure and document requirements.*

Use case modeling, including use case diagrams and use case textual specifications, is commonly applied to structure and document requirements. Use Case Specifications (UCS) are usually textual documents complying with a use case template that, though helping read and review use cases, inevitably contains ambiguities. RUCM is a use case modeling approach, and its goal is to restrict the way users can document UCSs in order to reduce ambiguity, improve the understandability of use case models, and facilitate automated analysis to derive initial analysis models, which in the Unified Modeling Language (UML) [15] are minimally composed of class and interaction diagrams, and possibly other types of diagrams and constraints.

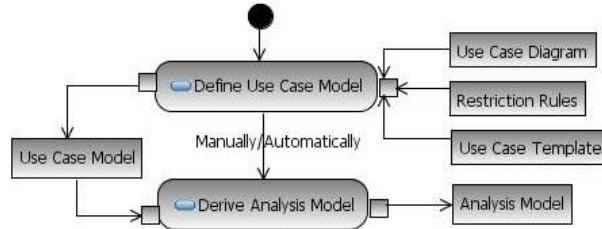


Figure 1 Defining use case model and deriving analysis model activities

The restriction rules and the use case template we specify should be applied during the requirements elicitation phase of use case-driven software development (e.g., [5]) to produce, to the extent possible, precise and unambiguous use case models, which is modeled as the first activity `Define Use Case Model` of the activity diagrams shown in Figure 1. Our use case modeling approach (RUCM) takes in input the use case diagram, the restriction rules, and the use case template. A use case diagram in UML [18], is used to represent relationships among actors and use cases; the restriction rules we defined are applied to restrict the way users can write UCSs; the use case template is the means to structure UCSs. With a use case model documented by applying RUCM, initial analysis models can then be derived as denoted by the activity `Derive Analysis Model`. This step is usually manually performed by system analysts but can also be automated, especially if inputted use case models are less ambiguous and thus automated analysis can be facilitated. In this document, we report on how to apply RUCM to define use case model (the first activity of Figure 1).

In this rest of the section, we first describe RUCM including the use case template and the restriction rules. Then we use a running example to illustrate how to define use case models by applying RUCM.

# RUCM Use Case Template

A use case template is used to structure use case specifications. Our RUCM template is based in part on the results of a thorough literature review and the need to devise transformation rules to analysis models.

**R**UCM template integrates elements of existing related works and it contains fields that are commonly encountered in conventional templates, but seeks to better specify the structure of flows of events of a use case specification. RUCM template has eleven first-level fields (first column of Table 1). The last four fields are decomposed into second-level fields (second column of the last four rows).

## **Use Case Name**

This field gives the name of the use case. It usually starts with a verb (e.g., *Withdraw Fund*) and should be consistent with the name of the same use case in the use case diagram.

## **Brief Description**

This field summarizes the use case in a short paragraph. It captures the essence of the use case.

## **Precondition**

The precondition of a use case specifies what must be always true before the use case begins.

## **Primary Actor**

The primary actor of a use case is the principle actor which initiates the use case.

## **Secondary Actors**

The secondary actors of a use case are the actors that the system relies on to accomplish the use case.

## **Dependency**

This field specifies <<include>> and <<extend>> relationships of a use case to other use cases.

## **Generalization**

This field specifies generalization relationships a use case to other use cases.

## **Basic Flow**

The basic flow of a use case describes a main successful path that satisfies the interests of the stakeholders. It often does not include any condition or branching. It is recommended to describe separately the conditions and branching in alternative flows. A basic flow is composed of a sequence of steps and a postcondition. Each UCS can only have one basic flow.

The action steps can be one of the following five interactions:

- 1) Primary actor → system: the primary actor sends a request and data to the system;
- 2) System → system: the system validates a request and data;
- 3) System → system: the system alters its internal state (e.g., recording or modifying something);
- 4) System → primary actor: the system replies to the primary actor with a result;
- 5) System → secondary actor: the system sends requests to a secondary actor.

All steps are numbered sequentially. This implies that each step is completed before the next one is started. If there is a need to express conditions, iterations, or concurrency, then specific keywords, specified as restriction rules should be applied.

## **Alternative Flows**

Alternative flows describe all the other scenarios or branches, both success and failure. An alternative flow always depends on a condition occurring in a specific step in a flow of reference, referred to as *reference flow*, and that reference flow is either the basic flow or an alternative flow itself. The branching condition is

specified in the reference flow by following restriction rules (R20 and R22). We refer to steps specifying such conditions as *condition steps* and the other steps as *action steps*. Similarly to the basic flow, an alternative flow is composed of a sequence of numbered steps.

We classify alternative flows into three types:

- 1) A *specific alternative flow* is an alternative flow that refers to a specific step in the reference flow.
- 2) A *bounded alternative flow* is a flow that refers to more than one step in the reference flow—consecutive steps or not.
- 3) A *global alternative flow* (called *general alternative flow* in [3]) is an alternative flow that refers to any step in the reference flow.

Distinguishing different types of alternative flows makes interactions between the reference flow and its alternative flows much clearer. For specific and bounded alternative flows, a RFS (Reference Flow Step) section, specified as rule R19, is used to specify one or more (reference flow) step numbers. Whether and where the flow merges back to the reference flow or terminates the use case must be specified as the last step of the alternative flow. Similarly to the branching condition, merging and termination are specified by following restriction rules (R24 and R25—Section Restriction rules on the use of keywords for specifying control structures (R17-R25) and R26). By doing so, we can avoid potential ambiguity in UCSs caused by unclear specification of interactions between the basic flow and its corresponding alternative flows.

Each alternative flow must have a postcondition. It is usual to provide a postcondition describing a constraint that must be true when a use case terminates. If the use case contains alternative flows, then the postcondition of the use case should describe not only what must be true when the basic flow terminates but also what must be true when each alternative flow terminates. The branching condition to each alternative flow is then necessarily part of the postcondition (to distinguish the different possible results). In such a case, the postcondition becomes complex and the branching condition for each alternative flow is redundantly described (both in the steps of flows and the postcondition), which therefore increases the risk of ambiguity in UCSs. Our template enforces that each flow of events (both basic flow and alternative flows) of a UCS contains its own postcondition and therefore avoids such ambiguity.

Table 1 Use case template

<b>Use Case Name</b>	The name of the use case. It usually starts with a verb.
<b>Brief Description</b>	Summarizes the use case in a short paragraph.
<b>Precondition</b>	What should be true before the use case is executed.
<b>Primary Actor</b>	The actor which initiates the use case.
<b>Secondary Actors</b>	Other actors the system relies on to accomplish the services of the use case.
<b>Dependency</b>	Include and extend relationships to other use cases.
<b>Generalization</b>	Generalization relationships to other use cases.
<b>Basic Flow</b>	Specifies the main successful path, also called “happy path”.
<b>Steps (numbered)</b>	Flow of events.
<b>Postcondition</b>	What should be true after the basic flow executes.
<b>Specific Alternative Flows</b>	Applies to one specific step of the basic flow.
<b>RFS</b>	A reference flow step number where flow branches from.
<b>Steps (numbered)</b>	Flow of events.
<b>Postcondition</b>	What should be true after the alternative flow executes.
<b>Global Alternative Flows</b>	Applies to all the steps of the basic flow.
<b>Steps (numbered)</b>	Flow of events.
<b>Postcondition</b>	What should be true after the alternative flow executes.
<b>Bounded Alternative Flows</b>	Applies to more than one step of the basic flow, but not all of them.
<b>RFS</b>	A list of reference flow steps where flow branches from.
<b>Steps (numbered)</b>	Flow of events.
<b>Postcondition</b>	What should be true after the alternative flow executes.

## RUCM Restriction Rules

*RUCM contains 26 restriction rules to restrict the way that users write use case specifications. They are based on a thorough literature review and have been experimentally evaluated to be understandable, easy to apply, and not restrictive. Together with the RUCM template, we can expect more precise and comprehensible use case models.*

**R**UCM restriction rules are classified into two groups: restrictions on the use of natural language, and restrictions enforcing the use of specific keywords for specifying control structures. The first group of restrictions is further divided into two categories according to their location of application (see below). Each restriction rule is assigned a unique number.

### **Restriction rules on the use of natural language and apply only to action steps (R1-R7)**

Seven restriction rules (R1-R7) constrain the use of natural language (Table 2): the table explains why they are needed to reduce ambiguity. Notice that these rules apply only to action steps; they do not apply to condition steps or preconditions or postconditions.

Table 2 Restrictions (R1-R7)

#	Description	Explanation
R1	The subject of a sentence in basic and alternative flows should be the system or an actor.	Enforce describing flows of events correctly. These rules conform to our use case template (the five interactions).
R2	Describe the flow of events sequentially.	
R3	Actor-to-actor interactions are not allowed.	
R4	Describe one action per sentence. (Avoid compound predicates.)	Otherwise it is hard to decide the sequence of multiple actions in a sentence.
R5	Use present tense only.	Enforce describing what the system does, rather than what it will do or what it has done.
R6	Use active voice rather than passive voice.	Enforce explicitly showing the subject and/or object(s) of a sentence.
R7	Clearly describe the interaction between the system and actors without omitting its sender and receiver.	

### **Restriction rules on the use of natural language and apply to all sentences (R8-R16)**

Nine restriction rules R8-R16 constraining the use of natural language are presented in Table 3. They apply to all sentences in a UCS: action steps, condition steps, preconditions, postconditions, and sentences in the brief description.

Rules R8-R10 and R16 are to reduce ambiguity of UCSs; the remaining rules are specifically to facilitate automated generation of analysis models, though they can also help reduce ambiguity. These two sets of restrictions are thought to be good practice for writing clear and concise UCSs (e.g., [4, 6, 22]) except for R13 and R15. These two rules are proposed in this work because we perceived that negative adverbs, negative adjectives, and participle phrases are very difficult to parse for natural language parsers. R9 requires using words consistently to document UCSs. A common approach to do so is to use a domain model and glossary (e.g., [16], [5]) as a basis to write UCSs.

Table 3 Restrictions (R8-R16)

#	Description	Explanation
R8	Use declarative sentence only. “Is the system idle?” is a non-declarative sentence.	Commonly required for writing UCSSs.
R9	Use words in a consistent way.	Keep one term to describe one thing.
R10	Don’t use modal verbs (e.g., <i>might</i> )	Modal verbs and adverbs usually indicate uncertainty; therefore metrics should be used if possible.
R11	Avoid adverbs (e.g., <i>very</i> ).	
R12	Use simple sentences only. A simple sentence must contain only one subject and one predicate.	Facilitate automated NL parsing and reduce ambiguity.
R13	Don’t use negative adverb and adjective (e.g., <i>hardly</i> , <i>never</i> ), but it is allowed to use <i>not</i> or <i>no</i> .	
R14	Don’t use pronouns (e.g., <i>he</i> , <i>this</i> )	
R15	Don’t use participle phrases as adverbial modifier. For example, the italic-font part of the sentence “ATM is idle, <i>displaying a Welcome message</i> ”, is a participle phrase.	
R16	Use “the system” to refer to the system under design consistently.	Keep one term to describe the system; therefore reduce ambiguity.

### **Restriction rules on the use of keywords for specifying control structures (R17-R25) and R26**

The remaining ten restriction rules (Table 4) constrain the use of control structures, except R26 that specifies that each basic flow and alternative flow should have its own postcondition. R17 and R18 specify keywords to describe use case dependencies include and extend. Sentences containing the keywords INCLUDE USE CASE and EXTENDED BY USE CASE are referred to as dependency sentences. R19 specifies keyword RFS, which is used in a specific (or bounded) alternative flow to refer to a step number (or a set of step numbers) of a reference flow step that this alternative flow branches from.

Rules R20-R23 specify the keywords used to specify conditional logic sentences (IF-THEN-ELSE-ELSEIF-ENDIF), concurrency sentences (MEANWHILE), condition checking sentences (VALIDATES THAT), and iteration sentences (DO-UNTIL), respectively. The keyword IF-THEN-ELSE-ELSEIF-ENDIF can be used in three different ways (these are specified as a grammar): 1) IF-THEN-ENDIF (appears in one flow only), 2) IF-THEN-ELSE-ENDIF (everything is in one flow or IF-THEN is used in basic flows; while ELSE is used in alternative flows.), and 3) IF-THEN-ELSEIF-THEN-ENDIF (everything is in one flow or IF-THEN is used in basic flows; while ELSEIF-THEN-ENDIF is used in alternative flows.). Keyword VALIDATES THAT (R22) means that the condition is evaluated by the system and must be true to proceed to the next step. This rule also requires an alternative flow describing what happens when the validation fails (the condition does not hold). Rules R20 and R22 are two complex and composite rules, when compared with the others of the same rule set, because both of them require that UCS designers look at multiple steps in two different flows: the basic flow and an alternative flow.

R24 and R25 specify keywords ABORT and RESUME STEP to describe an exceptional exit action and when an alternative flow goes back to its corresponding basic flow, respectively. These two rules also specify that an alternative flow ends either with ABORT or RESUME STEP, which means that the last step of the alternative flow should clearly specify whether the flow returns back to the basic flow and where (using keywords RESUME STEP followed by a returning step number) or terminates (using keyword ABORT).

R17-R21 and R23 have been proposed in the literature and we reused them with some variation. R22, R24 and R25 are newly proposed in this work for the purpose of making the whole set of restrictions as complete as possible so that flows of events and interactions between the basic flow and the alternatives can be clearly and concisely specified. Applying this set of rules facilitates automated NL processing (e.g., correctly parse sentences with our specified keywords) and generating of analysis models, especially sequence diagrams which also helps reducing ambiguity of UCSSs.

Table 4 Restrictions (R17-R26)

#	Description	#	Description
R17	INCLUDE USE CASE	R22	VALIDATE THAT
R18	EXTENDED BY USE CASE	R23	DO-UNTIL
R19	RFS	R24	ABORT
R20	IF-THEN-ELSE-ELSEIF-ENDIF	R25	RESUME STEP
R21	MEANWHILE	R26	Each basic flow and alternative flow should have their own postconditions.

## Example of Use – Withdraw Fund

*An example is worth a thousand of words.*

**A**n example of use case descriptions documented with RUCM is presented in Table 5 and Table 6. The original design of the use case description is from [9]. We rewrote it by applying RUCM. As shown in Table 5 and Table 6, the use case *Withdraw Fund* contains one basic flow, one specific alternative flow, one bounded alternative flow, and one global alternative flow. The specific and bounded alternatives correspond to four basic steps containing the keyword VALIDATES THAT. Notice that this is just one possible definition of the UCS applying our template and restrictions; it is possible to have different but equivalent definitions: for example, using the keyword IF-THEN-ELSE-ELSEIF-ENDIF instead of VALIDATES THAT.

Table 5 Use case Withdraw Fund (part 1)

Use Case Name	Withdraw Fund
Brief Description	ATM customer withdraws a specific amount of funds from a valid bank account.
Precondition	The system is idle. The system is displaying a Welcome message.
Primary Actor	ATM customer
Secondary Actors	None
Dependency	INCLUDE USE CASE Validate PIN.
Generalization	None
Basic Flow	<b>Steps</b> <b>1</b> INCLUDE USE CASE Validate PIN. <b>2</b> ATM customer selects Withdrawal through the system <b>3</b> ATM customer enters the withdrawal amount through the system. <b>4</b> ATM customer selects the account number through the system. <b>5</b> The system VALIDATES THAT the account number is valid. <b>6</b> The system VALIDATES THAT ATM customer has enough funds in the account. <b>7</b> The system VALIDATES THAT the withdrawal amount does not exceed the daily limit of the account. <b>8</b> The system VALIDATES THAT the ATM has enough funds. <b>9</b> The system dispenses the cash amount. <b>10</b> The system prints a receipt showing transaction number, transaction type, amount withdrawn, and account balance. <b>11</b> The system ejects the ATM card. <b>12</b> The system displays Welcome message. <b>Postcondition</b> ATM customer funds have been withdrawn.

Table 6 Use case Withdraw Fund (part 2)

Bounded Alternative Flows	RFS Basic Flow 5-7	
Global Alternative Flows	1	The system displays an apology message MEANWHILE the system ejects the ATM card.
	2	The system shuts down.
	3	ABORT.
	<b>Postcondition</b>	ATM customer funds have not been withdrawn. The system is shut down.
Specific Alternative Flows	IF ATM customer enters Cancel THEN	
	1	The system cancels the transaction MEANWHILE the system ejects the ATM card.
	2	ABORT.
	ENDIF	
	<b>Postcondition</b>	ATM customer funds have not been withdrawn. The system is idle. The system is displaying a Welcome message.
Specific Alternative Flows	RFS Basic Flow 8	
Global Alternative Flows	1	The system displays an apology message MEANWHILE the system ejects the ATM card.
	2	ABORT.
	<b>Postcondition</b>	ATM customer funds have not been withdrawn. The system is idle. The system is displaying a Welcome message.

In the rest of the section, we learn how RUCM restriction rules are applied based on the example by comparing with the original design of the use case *Withdraw Fund*.

#### PRE CONDITION:

Original sentence: ATM is idle, displaying a Welcome message.

Rewritten sentence: The system is idle. The system is displaying a Welcome message.

Reason: The original sentence breaks R15, which suggests not using participle phrases as adverbial modifier. Therefore, the original sentence is split into two simple sentence.

#### BASIC FLOW STEP 1

Original sentence: Include Validate PIN abstract use case

Rewritten sentence: Basic Flow Step 1

Reason: the keyword INCLUDE USE CASE specified as R17, should be used to describe the dependency relationship between use cases *Withdraw Fund* and *Validate PIN*.

#### BASIC FLOW STEP 2-4

Original sentence: Customer selects Withdrawal, enters the amount, and selects the account number.

Rewritten sentences: Basic Flow Step 2, Step 3, and Step 4

Reason: the original sentence is not a simple sentence (violating R12 and R4); it contains three predicates, which implies three action steps. Besides, it is not clear whether these three actions occur concurrently or sequentially. Therefore, we rewrite the sentence into three sequential action steps: steps 2-4.

As specified in R9, the terms should be used in a consistent way; therefore “ATM customer” should be used instead of “Customer: in the original sentence.

R7 says that the interaction between actors and the system should be clearly described without omitting its sender and receiver. The original sentence violates R7 and we rewrite the sentence by explicitly saying “...through the system”.

#### BASIC FLOW STEP 5-9 + ALTERNATIVE FLOWS

Original sentences: Basic flow:

- 1) System checks whether customer has enough funds in the account and whether the daily limit will not be exceeded.
- 2) If all checks are successful, system authorizes dispensing of cash.

Alternative flows:

- 1) If the system determines that the account number is invalid, it displays an error message and ejects the card.
- 2) If the system determines that there are insufficient funds in the customer's account, it displays an apology and ejects the card.
- 3) If the system determines that the maximum allowable daily withdrawal amount has been exceeded, it displays an apology and ejects the card.
- 4) If the ATM is out of funds, the system displays an apology, ejects the card, and shuts down the ATM.

Rewritten sentences: Basic Flow Step 5, Step 6, Step 7, and Step 8

Reason: R22 specifies the keyword VALIDATES THAT, which means that the condition is evaluated by the system and must be true to proceed to the next step. We use this keyword to rewrite all the original sentences. Step 1 of the original basic flow contains two condition checking sentences and the problem is that it is not clear with condition checking action should be taken first or both of them occur concurrently. By using the keyword "VALIDATES THAT", there is no need to have the "if" condition of Step 2 of the original basic flow and therefore the ambiguity is avoided and the description is simplified.

The first original alternative flow contains an "if" condition which should be described as a condition check action sentence in the basic flow (Basic Flow Step 5 of the rewritten use case) by using the keyword VALIDATES THAT. The same rule is applied to the other three alternative flows of the original use case specification.

The alternative flows of the original use case specification do not clearly specify the locations where they branched from the basic flow. The keyword RFS should be applied to clearly state the location (step) that an alternative branches from its reference flow.

It is also important to distinguish different types of alternative flows.

The basic flow and each alternative flow should have its own postcondition.

Each alternative flow must end either with ABORT or RESUME STEP.

## Quick Reference Tables

The quick reference tables available here provide a way for users quickly look at a specific restriction rule including its definition, explanation, and example.

Table 7 Restriction rules R1-R7

#	Description	Example	
		RIGHT	WRONG
<b>0</b>	A step is either one simple sentence (see below) or one complex sentence (with keywords IF, WHILE, VALIDATES, DO—see below).	N/A	N/A
<b>R1</b>	The subject of a sentence in basic and alternative flows should be the system or an actor.	<i>The system</i> ejects the ATM card	<i>The card</i> has been ejected
<b>R2</b>	Describe the flow of events sequentially.	1. The system dispenses the cash amount. 2. The system ejects the ATM card.	1. The system ejects the ATM card. 2. The system dispenses the cash amount.
<b>R3</b>	Actor-to-actor interactions are not allowed.	The customer inserts the ATM card into the card reader.	The customer gives the teller the ATM card.
<b>R4</b>	Describe one action per sentence. (Avoid compound predicates.)	the system cancels the transaction MEANWHILE the system ejects the card. or 1. the system cancels the transaction. 2. the system ejects the card	...the system cancels the transaction and ejects the card.
<b>R5</b>	Use present tense only.	The system <i>ejects</i> the card.	The system <i>ejected</i> the card.
<b>R6</b>	Use active voice rather than passive voice.	The system <i>ejects</i> the card.	The card <i>is ejected</i> .
<b>R7</b>	Clearly describe the interaction between the system and actors without omitting its sender and receiver.	ATM customer enters PIN number <i>to the system</i> .	Customer enters PIN.

Table 8 Restriction rules R8-R16

#	Description	Example	
		RIGHT	WRONG
<b>R8</b>	Use declarative sentence only. “Is the system idle?” is a non-declarative sentence.	The system ejects the card.	Ejects the card.
<b>R9</b>	Use words in a consistent way.	<i>ATM customer</i> inserts the ATM card...	<i>Customer</i> inserts the ATM card...
<b>R10</b>	Don’t use modal verbs (e.g., <i>might</i> )	The system ejects the card.	The system <i>might</i> eject the card.
<b>R11</b>	Avoid adverbs (e.g., <i>very</i> ).	The system ejects the card.	The system <i>very likely</i> ejects the card.
<b>R12</b>	Use simple sentences only. A simple sentence must contain only one subject and one predicate.	1. The system displays ATM customer accounts. 2. The system prompts ATM customer for ...	System displays customer accounts <i>and</i> prompts customer for transaction type...
<b>R13</b>	Don’t use negative adverb and adjective (e.g., <i>hardly</i> , <i>never</i> ), but it is allowed to use <i>not</i> or <i>no</i> .	The PIN number <i>has not</i> been validated	The PIN number <i>has never been</i> validated.
<b>R14</b>	Don’t use pronouns (e.g. <i>he</i> , <i>this</i> )	<i>...the system</i> reads the card number.	<i>...it</i> reads the card number.
<b>R15</b>	Don’t use participle phrases as adverbial modifier.  For example, the italic-font part of the sentence “ATM is idle, <i>displaying a Welcome message</i> ”, is a participle phrase.	The system is idle. The system is displaying a Welcome message.	ATM is idle, displaying a Welcome message.
<b>R16</b>	Use declarative sentence only. “Is the system idle?” is a non-declarative sentence.	the system	“ATM” or “The ATM system”

Table 9 Restriction rules R17-R20

#	Description	Grammar	Explanation	Example	
				RIGHT	WRONG
R17	Use keywords INCLUDE USE CASE to describe the include dependencies with other use cases.	INCLUDE USE CASE <included use case name>	The keywords can be used in basic and step alternative flows.	INCLUDE USE CASE Validate PIN	Include Validate PIN abstract use case.
R18	Use keywords EXTENDED BY USE CASE to refer to the extended use case.	EXTENDED BY USE CASE <extending use case> or <specific use case> Appears either in the action part of a THEN or ELSE in the main flow, or as an action in an alternative flow.	The keywords can be used in basic and alternative flows.	EXTENDED BY USE CASE CreateIncident	Use case CreateIncident extends the current use case.
R19	Use keyword RFS in a specific (or bounded) alternative flow to refer to a step number (or a lower bound step number and an upper bound step number) of a reference flow step that this alternative flow corresponds to.	RFS <reference flow step #> (specific alternative flow) RFS <reference flow step numbers> (bounded alternative flow) Not required notation for global alternative flow.	One specific or bounded alternative flow must correspond to exactly one or more than one reference flow steps.	RFS Basic Flow 5 ... RFS Basic Flow 5-7, 10, 14 ...	
R20	Use pairs of keywords of IF, THEN, ELSE, ELSEIF, and ENDIF to describe conditional logic sentences.	IF <condition> THEN <steps> ENDIF	Appears in one flow only.	IF the system recognizes the ATM card, THEN the system reads the ATM card number, ENDIF.	If the system recognizes the card, it reads the card number.
		IF <condition> THEN <steps> – ELSE <steps> ENDIF	IF-THEN is used in basic flows; while ELSE is used in alternative flows (see RFS). Or everything is used in only one flow.		
		IF <condition> THEN <steps> – ELSEIF <condition> THEN <steps> ENDIF	IF-THEN is used in basic flows; while ELSEIF-THEN is used in alternative flows (see RFS). Or everything is used in only one flow.		

Table 10 Restriction rules R21-R26

#	<b>Description</b>	<b>Grammar</b>	<b>Explanation</b>	<b>Example</b>	
				<b>RIGHT</b>	<b>WRONG</b>
<b>R21</b>	Use keyword MEANWHILE to describe concurrency.	<action> MEANWHILE <action>	It implies that the sentence before keyword MEANWHILE and the sentence after the keyword occur concurrently.	...the system cancels the transaction MEANWHILE the system ejects the card.	...the system cancels the transaction and ejects the card.
<b>R22</b>	Use keyword VALIDATES THAT to describe condition checking sentences. VALIDATES THAT means that the condition is evaluated and must be true to proceed to the next step.	VALIDATES THAT <condition>	The alternative case (the condition does not hold) must be described in its corresponding alternative flow (BFS).	...the system VALIDATES THAT the user-entered PIN...	... the system checks whether the user-entered PIN...
<b>R23</b>	Use keyword pair DO and UNTIL to describe iteration.	DO <steps> UNTIL <condition>	Following keyword DO is a sequence of steps. Following keyword UNTIL is a loop ending condition.	1. DO 2. action1 3. action2 4. UNTIL condition	
<b>R24</b>	Use keyword ABORT to describe an exceptionally exit action. An alternative flow ends either with ABORT or RESUME STEP.	ABORT	Used in alternative flows, iterative, and conditional logic sentences. It means the ending of a use case.		
<b>R25</b>	Use keyword pair RESUME STEP to describe the situation where an alternative flow goes back to its corresponding basic flow. An alternative flow ends either with ABORT or RESUME STEP.	RESUME STEP <basic flow step #>	Used in alternative flows.	RESUME STEP 5	
<b>R26</b>	Each basic flow and alternative flow should have their own postconditions.	Refer to restriction R19.			

附录 D 测试报告

# RUCM 规则检查测试报告

## 测试报告

测试版本号	测试人	测试时间	测试范围	备注
V1.0	梁保宇	2018/12/6	规则检查/文件载入	

# 目 录

1 概述 .....	3
2 测试目的 .....	3
3 需求实现度 .....	3
4 测试功能点 .....	3
5 测试环境 .....	错误!未定义书签。
6 测试结果统计.....	6
6.1 测试用例执行情况 .....	6
6.2 BUG 统计 .....	7
6.2.1 Bug 趋势图.....	错误!未定义书签。
6.2.2 所有 Bug 等级分布图.....	7
6.2.3 所有 Bug 所属模块分布图.....	8
6.2.4 遗留 Bug 统计.....	8
7 风险分析 .....	错误!未定义书签。
附：产品线自身上线标准.....	错误!未定义书签。

# 1 概述

本次测试的功能点主要在数据读取、规则检查的正确性。检查对象为读取类 Loader、规则类 Rule、DefaultRule17-26 中的各 check() 方法。

本次测试对应的开发版本为 2018 年 12 月 5 日完成版本。测试环境为 Windows 10 + Python3.6 环境，NLP 模块使用远程 NLP 处理器测试。

更新：已测试 2018 年 12 月 7 日版本，测试用例同上，主要目的为检查上次测试 BUG 修复情况。

# 2 测试目的

本文档为 RUCM 规则检查项目的规则检查功能的测试报告，从各个方面对测试对象、测试过程进行评估，得出版本质量结论和主要风险。

更新：第二次检查主要目的为检查上次测试 BUG 修复情况。

# 3 需求实现程度

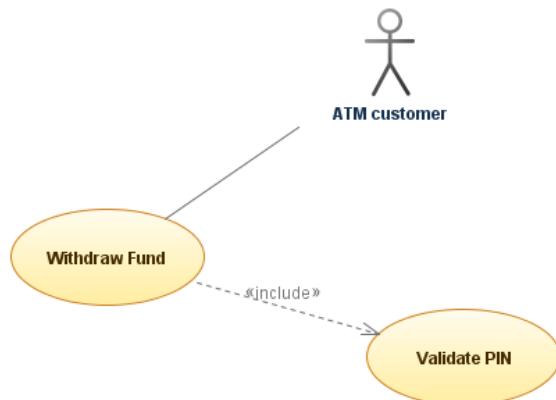
编号	需求描述	是否实现
1	RuleLoader 读取规则模板	是
2	RUCMLoader 读取 rucm 文件	是
3	对读取到的 rucm 文件进行 NLP 处理，填充 RUCM 相关各类	是
4	RuleLoader 生成规则，载入规则库	是
5	Rule 规则检查	是
6	生成错误报告	是

# 4 测试功能点

测试类型	测试模块	测试功能点
功能测试	Rule 规则检查	1、默认规则 1 测试
		3、默认规则 3 测试
		4、默认规则 4 测试

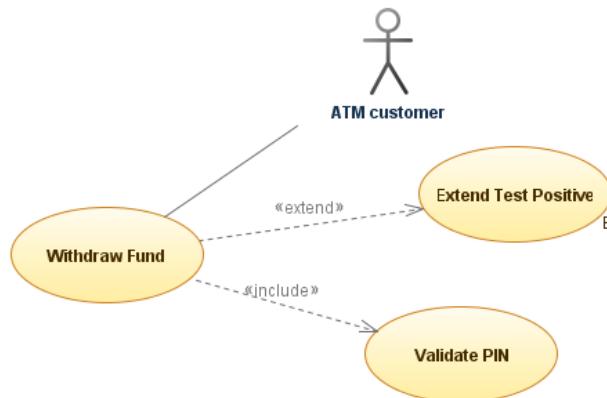
		5、默认规则 5 测试
		8、默认规则 8 测试
		10、默认规则 10 测试
		11、默认规则 11 测试
		12、默认规则 12 测试
		13、默认规则 13 测试
		15、默认规则 15 测试
		17、默认规则 17 测试
		18、默认规则 18 测试
		19、默认规则 19 测试
		20、默认规则 20 测试
		21、默认规则 21 测试
		22、默认规则 22 测试
		23、默认规则 23 测试
		24、默认规则 24 测试
		25、默认规则 25 测试
		26、默认规则 26 测试
	Loader 测试	正确样例测试

## 5 测试样例说明



图一、测试采用用例图

测试用例共22个文件，包括1个无违反RUCM规则的测试样例和21个违反RUCM规则的测试样例。测试样例的整体用例图如图一。Extend关键字规则检查（Default Rule 18）用例图如图二。



图二、Extend关键字规则采用用例图

正确测试用例来源为 *Restricted Use Case Modeling Approach User Manual* 中 *Example of Use – Withdraw Fund* 板块下的用例模板，含违反规则情况的测试用例来源为 *Quick Reference Tables* 板块下的各错误例子。其中，关键字规则的错误用例为自制用例，包含违反关键字规则的若干种情况。

对每个测试用例文件而言（尤其是关键字规则），为保证测试的完整性，针对每条规则可能有多条违反规则的测试用例。正确的测试用例 Withdraw Fund 内容见附录。所有测试用例文件放置在 test 文件夹中。

TestError_default1.rucm	2018/12/11 22:13	RUCM 文件	45 KB
TestError_default3.rucm	2018/12/11 22:13	RUCM 文件	45 KB
TestError_default4.rucm	2018/12/11 22:13	RUCM 文件	44 KB
TestError_default4_2.rucm	2018/12/11 22:13	RUCM 文件	45 KB
TestError_default5.rucm	2018/12/11 22:13	RUCM 文件	45 KB
TestError_default8.rucm	2018/12/11 22:13	RUCM 文件	45 KB
TestError_default10.rucm	2018/12/11 22:13	RUCM 文件	45 KB
TestError_default11.rucm	2018/12/11 22:13	RUCM 文件	45 KB
TestError_default12.rucm	2018/12/11 22:13	RUCM 文件	45 KB
TestError_default14.rucm	2018/12/11 22:13	RUCM 文件	44 KB
TestError_default15.rucm	2018/12/11 22:13	RUCM 文件	44 KB
TestError_default17.rucm	2018/12/11 22:13	RUCM 文件	45 KB
TestError_default18.rucm	2018/12/11 22:13	RUCM 文件	51 KB
TestError_default19.rucm	2018/12/11 22:13	RUCM 文件	45 KB
TestError_default20.rucm	2018/12/11 22:13	RUCM 文件	58 KB
TestError_default21.rucm	2018/12/11 22:13	RUCM 文件	44 KB
TestError_default22.rucm	2018/12/11 22:13	RUCM 文件	45 KB
TestError_default23.rucm	2018/12/11 22:13	RUCM 文件	49 KB
TestError_default24.rucm	2018/12/11 22:13	RUCM 文件	45 KB
TestError_default25.rucm	2018/12/11 22:13	RUCM 文件	46 KB
TestError_default26.rucm	2018/12/11 22:13	RUCM 文件	42 KB
testNormal.rucm	2018/12/11 22:13	RUCM 文件	45 KB

## 6 测试结果统计

测试人员：梁保宇

测试时间：2018 年 12 月 06 日——2018 年 12 月 07 日

测试时间：2018 年 12 月 08 日——2018 年 12 月 08 日

### 6.1 测试用例执行情况

测试用例总数	22			备注	本次迭代用例
执行的用例总数	22	执行情况	100%	备注	
通过的用例总数	14	通过率	64%	备注	本次迭代用例
上版本遗留 BUG	第一版本测试				
是否解决完毕	第二版本：部分解决完毕				
是否满足产品线自身上线标准	D 级 第二版本：B 级				

版本质量等级划分：

A 级：所有功能都已实现，发现的 bug 都解决。

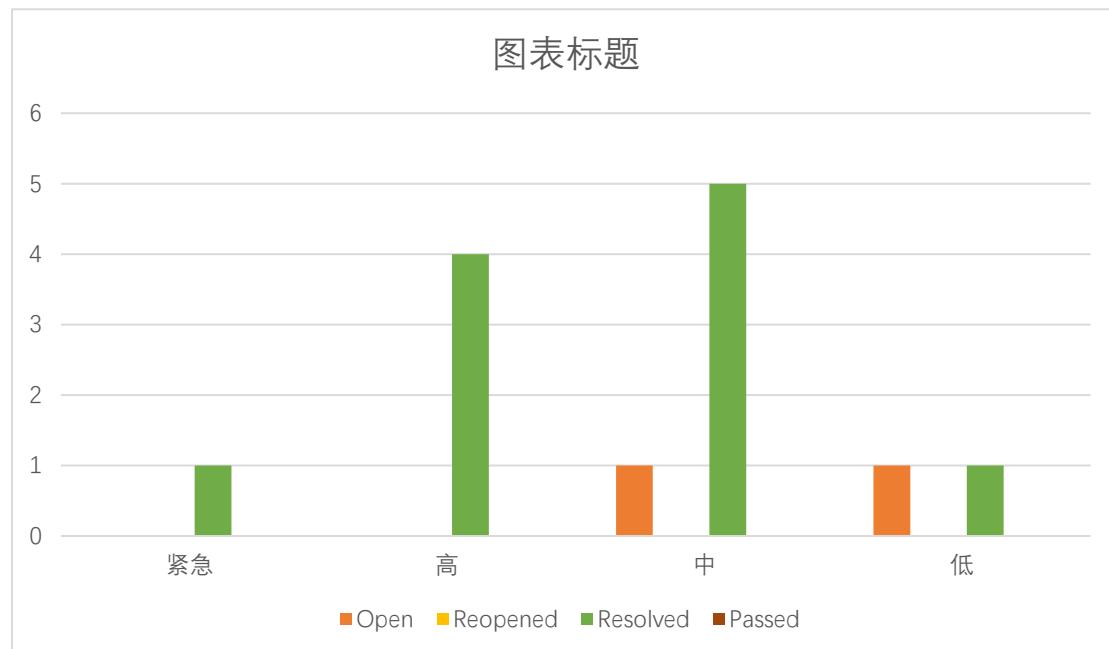
B 级：所有功能都已实现，还有遗留 bug，但是有规避措施，不影响用户使用。

C 级：主功能已实现，但存在严重 bug 未修复，有影响用户使用的可能。

D 级：主功能未完全实现，或存在非常严重的 bug 未修复，无法正常使用。

## 6.2 Bug 统计

### 6.2.1 所有 Bug 等级分布图

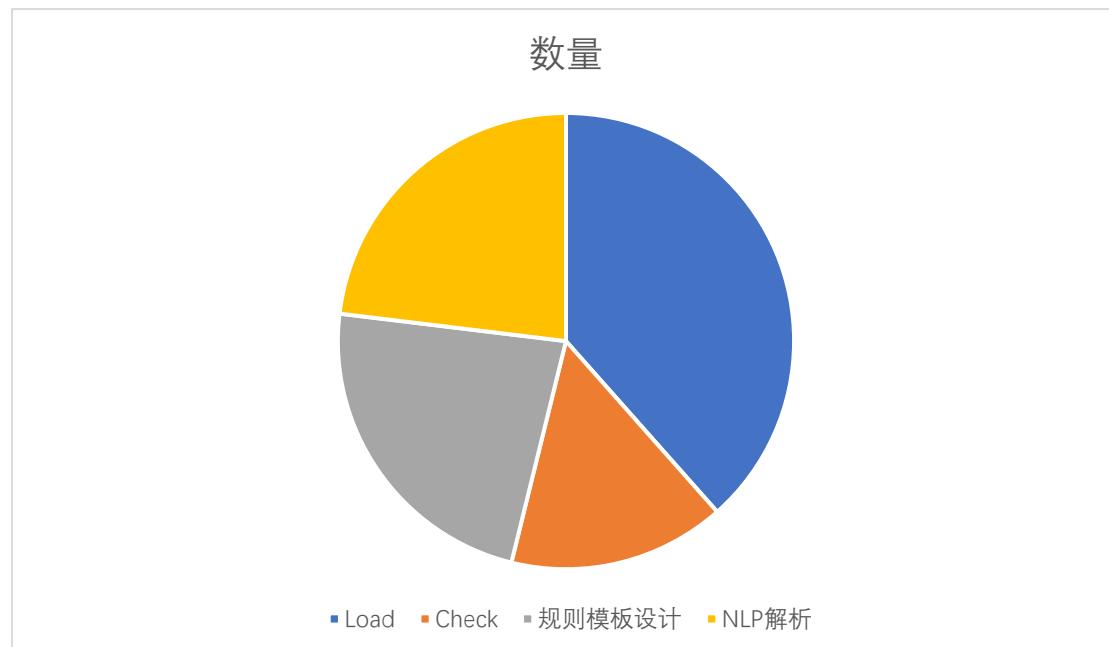


不同 status 下 Bug 严重等级分布表

severity	状态				
	Open	Reopened	Resolved	Passed	合计:
紧急	0	0	1	0	0
高	0	0	4	0	0
中	1	0	5	0	0
低	1	0	1	0	0
总计:	2	0	11	0	0

注：其中 Resolved 状态中包含 不可复现 和 转需求分析 状态。

### 6.2.2 所有 Bug 所属模块分布图



### 6.2.3 Bug 统计

序号	BUG 等级	内容	状态	出现机率	备注
L1	紧急	不指定规则文件时，程序直接崩溃	SOLVED	100%	本次迭代
L2	中	Global Alternative Flow 应没有 Rfs 字段，但是当前版本把 Global Alternative Flow 对应的 Condition 字段装入了 Rfs 字段，导致在获取 Sentence 时找不到 Global Alternative Flow 的对应的 Condition	SOLVED	出现 Global Alternative Flow 时 100%	本次迭代
L3	低	解析后 Sentence 中前后有多余的空格	SOLVED	100%	本次迭代
S1	中	规则模板，默认规则中第 7, 12 条规则出现没有宾语的情况报违反规则（例如：The system shuts down）	SOLVED	100%	本次迭代
S2	高	规则模板中第 8, 10, 11, 13, 14 条规则无法检测对应规则违反情况	SOLVED	100%	本次迭代

J1	高	从 sentence.words 里面取出来的是 map，而且只能取一次，取第二次就都取不出来了	SOLVED	100%	本次迭代
S3	高	规则模板中第三条检测不出错误用例	SOLVED	100%	本次迭代
C1	中	规则检查中含 Nature 的句子被过滤	SOLVED	100%	本次迭代
J2	中	现在分词无法辨析是用来做状语还是定语	OPEN	100%	本次迭代
L4	高	1-7 条规则检查只传入了 Basic Flow，没有传入 Alternative Flow	SOLVED	100%	本次迭代
J3	低	The system displays Welcome message. 中检测到两个动词导致违背简单句规则	OPEN	100%	本次迭代
C2	中	defaultrule20 产生的错误没有被加入错误库	SOLVED	100%	本次迭代
L5	中	加入嵌套 IF-THEN 语句后，INCLUDE USE CASE 语句取出来空列表	SOLVED	100%	本次迭代

# 7 附录

Use Case Specification	
Use Case Name	Withdraw Fund
Brief Description	ATM customer withdraws a specific amount of funds from a valid bank account.
Precondition	The system is idle. The system is displaying a Welcome message.
Primary Actor	None
Secondary Actors	None
Dependency	<b>INCLUDE USE CASE</b> Validate PIN
Generalization	None
Basic Flow	<p>Steps</p> <p>"Basic Flow" ▼</p> <p>1 <b>INCLUDE USE CASE</b> Validate PIN</p> <p>2 ATM customer selects Withdrawal through the system</p> <p>3 ATM customer enters the withdrawal amount through the system.</p> <p>4 ATM customer selects the account number through the system.</p> <p>5 The system <b>VALIDATES THAT</b> the account number is valid.</p> <p>6 The system <b>VALIDATES THAT</b> ATM customer has enough funds in the account.</p> <p>7 The system <b>VALIDATES THAT</b> the withdrawal amount does not exceed the daily limit of the account.</p> <p>8 The system <b>VALIDATES THAT</b> the ATM has enough funds.</p> <p>9 The system dispenses the cash amount.</p> <p>10 The system prints a receipt showing transaction number, transaction type, amount withdrawn, and account balance.</p> <p>11 The system ejects the ATM card.</p> <p>12 The system displays Welcome message.</p> <p>Postcondition ATM customer funds have been withdrawn.</p>
Bounded Alternative Flow (UnUsed) ▼	<p><b>RFS</b> Basic Flow 5-7</p> <p>1 The system displays an apology message <b>MEANWHILE</b> the system ejects the ATM card.</p> <p>2 The system shuts down.</p> <p>3 <b>ENDIF</b></p> <p>Postcondition ATM customer funds have not been withdrawn. The system is shut down.</p>
Global Alternative Flow (UnUsed) ▼	<p><b>IF</b> ATM customer enters Cancel <b>THEN</b></p> <p>1 The system cancels the transaction <b>MEANWHILE</b> the system ejects the ATM card.</p> <p>2 <b>ENDIF</b></p> <p>3 <b>ENDIF</b></p> <p>Postcondition ATM customer funds have not been withdrawn. The system is idle. The system is displaying a Welcome message.</p>
Specific	<b>RFS</b> Basic Flow 0

中 | 简 | 深 | 简