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

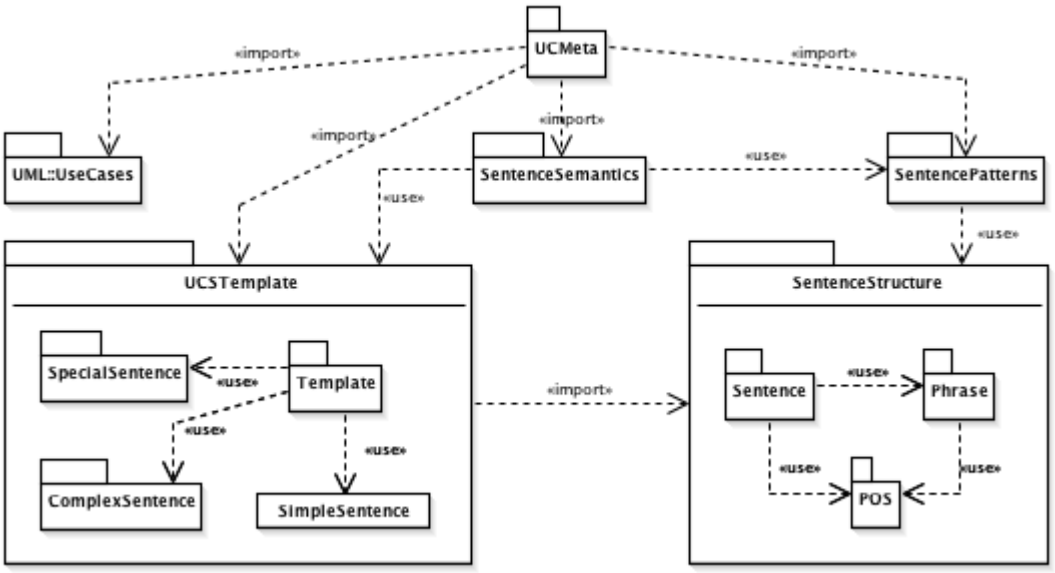
RUCM (Restricted Use Case Modeling)需 求 建 模 方 法是一种基于用例的方法，由结构化的用例规约模板和一组限 制规则构成。于 2013 年由 YUE T，BRIAND L C，LABICHE Y 等人提出结构化的模板定义了在用例规约中需要描述的 内容，如表 1 所列。限制规则包括自然语言规则与关键字。 实验证明，RUCM 方法可以有效地降低需求规约的二义性，同时又保留自然语言易理解和易使用的优点。

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
Alternative Flows	GuardField	引用流中的步骤号或守护条件
	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功能。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结果;

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

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

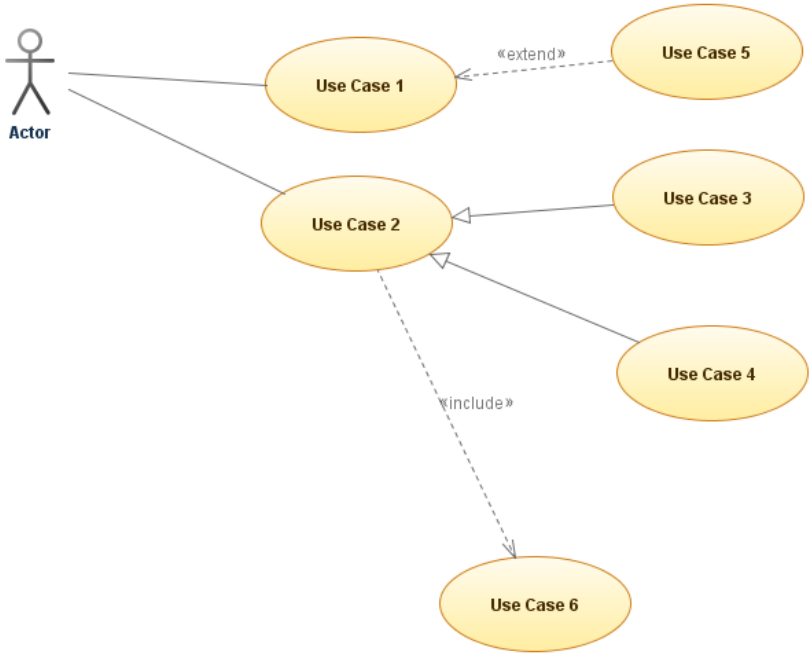
#	Description	Explanation
R8	Use declarative sentence only. "Is the system idle?" is a non-declarative sentence.	Commonly required for writing UCSs.
R9	Use words in a consistent way.	Keep one term to describe one thing.
R10	Don't use modal verbs (e.g., *might*)	Modal verbs and adverbs usually indicate uncertainty; therefore metrics should be used if possible.
R11	Avoid adverbs (e.g., *very*).	
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., *hardly*, *never*), but it is allowed to use *not* or *no*.	
R14	Don't use pronouns (e.g. *he*, *this*)	
R15	Don't use participle phrases as adverbial modifier. For example, the italic-font part of the sentence "ATM is idle, *displaying a Welcome message*", 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)

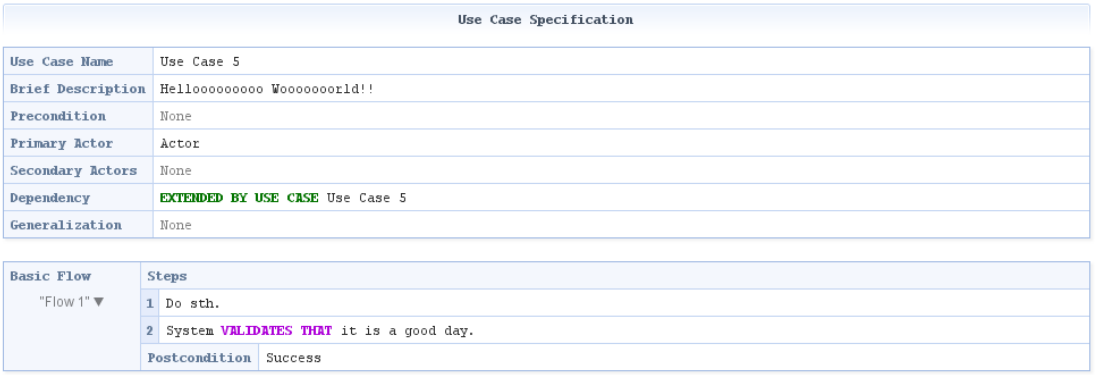
#	Description	#	Description
R17	INCLUDE USE CASE	R22	VALIDATE THAT
R18	EXTENDED BY USE CASE	R23	DO-UNTIL
R19	RFS	R24	ABORT
R20	IF-THEN-ELSE-ELSEIF-ENDIF	R25	RESUME STEP
R21	MEANWHILE	R26	Each basic flow and alternative flow should have their own postconditions.

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。以如下用例图为例。



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	Hellooooooooo Wooooooorld!!
Precondition	None
Primary Actor	Actor
Secondary Actors	None
Dependency	EXTENDED BY USE CASE Use Case 5
Generalization	None

Basic Flow	Steps	
"Flow 1" ▼	1	Do sth.
	2	System VALIDATES THAT it is a good day.
	Postcondition	Success

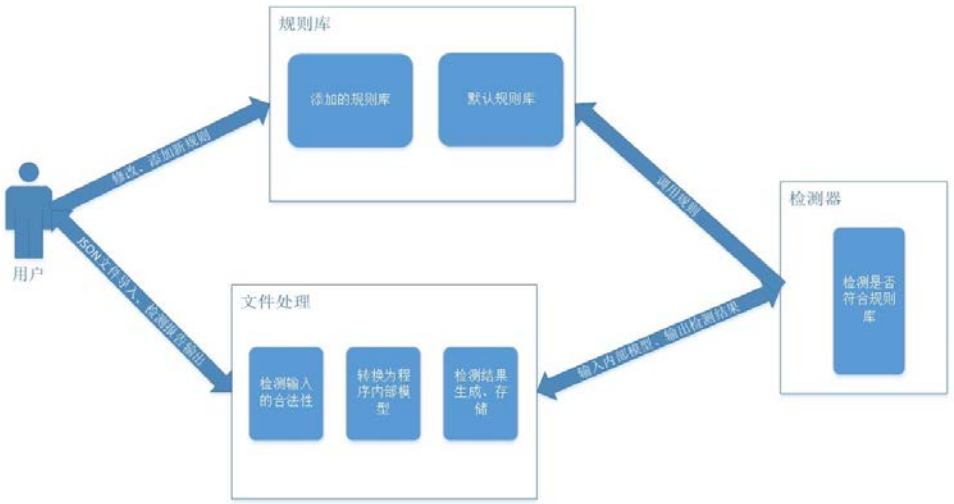
最后生成的 JSON 文件结构分别对模型中的元素、模型描述、模型名称、模型版本、引用关系、模型图组成进行了表示。表示内容如下。

表示内容	表示结构
模型元素——Actor	- content - description //Actor 描述 - name //Actor 名称 - type:"Actor"
模型元素——Use Case	- content - description //Use Case 描述 - extend/include //Use Case 的 extend/include 关系 - name //Use Case 名称 - specification //模板内容 -type:"Use Case"
模型元素——连线	- content - use case/actor //连线两端的对

	象 - description //连线描述 - name //连线名称 -type:"Relationship/Generalization"
模型描述	- description - content //对模型的描述 - type //content 的类型（一般为 String）
模型名称	- name - content //模型名称 - type //content 的类型（一般为 String）
模型图	-diagrams - nodes //图中的节点（用例、参与者） - links //图中的连线 - ucModel //所属模型 - top/bottom/width/height //位置信息

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。

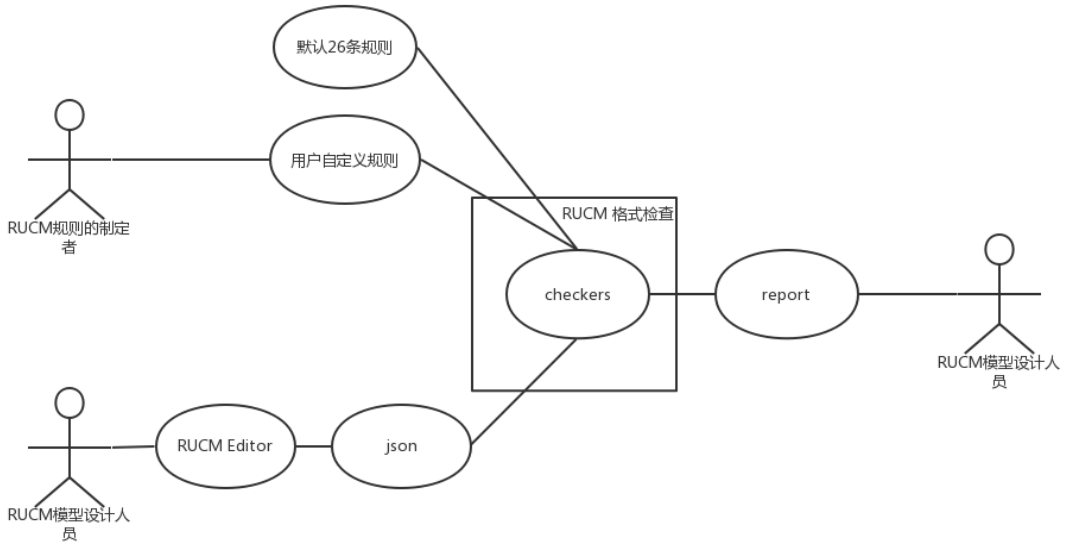
检测器模块为程序的核心实现。检测器结合规则库对转换后的程序内部模型进行检测，包括对 26 条默认规则的检测和对用户新添加规则的检测。并且生成检测报告，提交给文件处理模块，最终输出给用户。

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

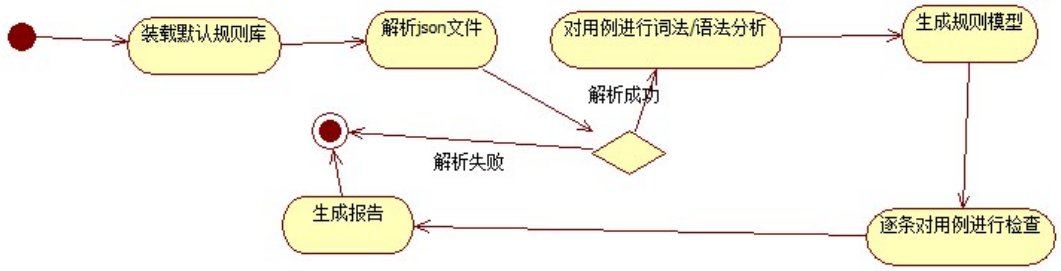
后续使用。系统主要接收、处理的数据为由 RUCM 建模软件产生的 JSON 数据和系统本身需要管理维护的规则库 R。该 JSON 数据主要是 RUCM 模型中的用例信息。每一用例信息由用例规约模板中的各字段组成，包括用例名、用例简介、前置条件、主要参与者、次要参与者、与其他用例的依赖、泛化关系、事件流等。同时，系统能够接收规则制定者根据需求对约束规则的添加、修改、删除。系统产生的输出为规则检查报告，该报告主要用于对 RUCM 建模人员所提交的 JSON 数据中违反限制规则的情况进行反馈，包括违反的限制规则编号、违反限制规则的用例名称等。另外，系统为需求建模人员提供可视化操作界面，用来展示现有规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

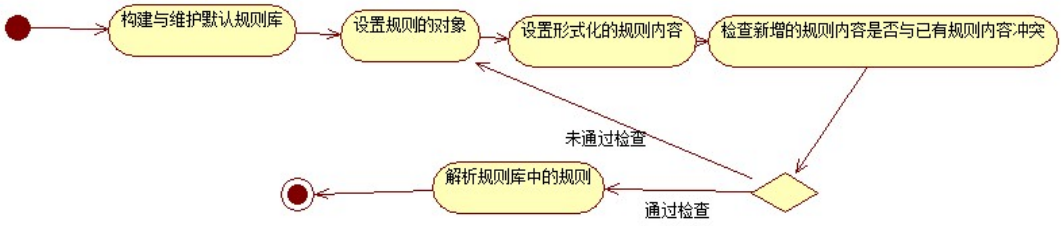


该系统以 checkers 为核心，checkers 以规则和 RUCM 模型的 json 文件为输入。其中，规则根据来源可以分为 2 种，用户自定义规则和默认 26 条规则。用户自定义规则由用户产生。这里的用户主要指的是 RUCM 规则的制定者。默认 26 条规则在第 3 部分有详细介绍，另外所有的 26 条规则和更详细的介绍和示例都可以在 RUCM manual 中找到。RCUM 默认设计人员使用 RUCM Editor 绘制编辑 RUCM 模型，之后可以用 Editor 的导出功能以 json 格式导出 RUCM 模型。该 json 文件作为 checkers 的另一输入，也就是该系统需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，同时也可以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，探索研究在不同干的规则约束条件下 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。