使用Uppaal进行多处理器实时系统可调度分析的工具原型——模型转换和结果解释

需求概述

**用户目的：**验证某一系统的可调度性

**问题：**如果自己使用某个模型检测工具，需要自己学习使用模型检测工具，需要自己抽象可调度性问题，验证结果的正确性极大程度依赖于系统模型建立的准确性和完整性。

**解决方法：用户只需要描述系统和任务，不需要再次从零开始建模，工具就能替用户建立模型、执行验证，并向用户展示验证结果。**

1. 用户输入对系统的描述，包括：

**Application:** Task (name), and its running period (number), offset (number)

**Dependency:** dependency between tasks

**Platform:** processing elements and its schedule strategy, bus and its strategy (name)

**Mapping:** which task run on which processing element (name)

**Characteristic:** best case execute time (number) and worst case execute time (number) of each mapping

**Property:** Schedule?

1. 用户选择执行验证功能
2. 用户得到：可调度性分析的结果：可调度 or 不可调度，显示一个不可调度的反例

概要设计：

模块和接口设计：

GUI: 包括一个主界面MainFrame，其中包含用户输入系统信息的SystemDescribe模块、用户选择验证的模块、结果显示ResultDisplay模块。

UserInput模块：处理用户的输入数据，包括记录任务属性、系统属性、统计任务数目、计算依赖数据传输时间等。

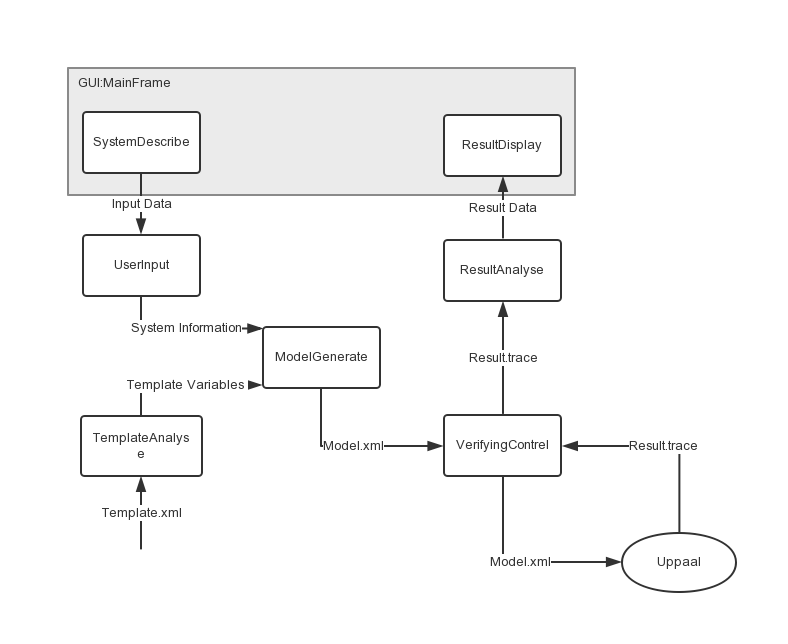
TemplateAnalyse模块:解析模板Template.xml，分析需要实例化的变量。

ModelGenerate模块:将用户输入的系统信息赋值给模板中响应的变量，生成Model.xml文件，即可用于验证的模型。

VerificationControl:控制在后台调用Uppaal验证Model.xml，并将验证结果输出到Result.trace中。是连接工具和Uppaal的模块。

ResultAnalysis:分析验证结果文件Result.trace，将分析的结果数据传输给结果显示模块。

各模块的关系如下图所示：

详细设计：

**Figure 1 module interaction**

类设计：

#界面类，包含要展示的元素,包括输入部分、功能控制部分、结果展示部分

class MainFrame

#method

self.show() #显示界面

showResult(Result result) #显示结果部分的界面

#获取用户的输入信息

class UserInput

#method:

getTask() #

get

#模板类，处理模板文件,识别一些标签

class Template

#method

readTemplate(path) #读template.xml文件

#模型类，处理模型文件

class Model

#method

model(UserInput uData,Template template) #生成model,把输入数据和模板结合起来

path modelGen(Model model) #生成model.xml

#结果类，主要进行结果分析

class Result

#method

#控制类，连接工具和Uppaal，控制uppaal执行

class Controller

#method

path modelValidate(path) #调用uppaal，验证model.xml,生成结果文件result.trace