### 2.3.4 MVC

MVC模式（Model-View-Controller）是软件工程中的一种软件架构模式，把软件系统分为三个基本部分：模型（Model）、视图（View）和控制器（Controller）。MVC是一种框架式软件设计典范，旨在用这种设计分离业务逻辑和数据现实，在界面和用户的交互被改进或定制的时候不需要影响到业务逻辑，实现一种设计上的解耦。

* （控制器 Controller）- 负责转发请求，对请求进行处理。
* （视图 View） - 图形界面。
* （模型 Model） - 程序应有的功能（实现算法等等）、数据管理和数据库设计(可以实现具体的功能)。



图 2.7 MVC模式

MVC有几大重要优点，对本项目的设计和代码编写十分有力，因此本项目在项目架构设计中选择了MVC架构。

### 2.3.5 java swing

Swing是一个为Java设计的GUI工具包。Swing是JAVA基础类的一部分。Swing包括了图形用户界面（GUI）器件如：文本框，按钮，分隔窗格和表。

Swing提供许多比AWT更好的屏幕显示元素。它们用纯Java写成，所以同Java本身一样可以跨平台运行，这一点不像AWT。它们是JFC的一部分。它们支持可更换的面板和主题（各种操作系统默认的特有主题），然而不是真的使用原生平台提供的设备，而是仅仅在表面上模仿它们。这意味着你可以在任意平台上使用JAVA支持的任意面板。轻量级组件的缺点则是执行速度较慢，优点就是可以在所有平台上采用统一的行为。

本项目采用java Swing进行界面设计和开发。

## 2.3 本章小结

本章主要关于是基于场景的并发模型存在一致性检验工具的总体概述，对相关的概念，工具以及使用的技术做一介绍。在后面的概要设计以及详细模块设计的部分，会再次引用本章节的相关知识。

# 第三章 预处理模块的设计与实现

## 3.1 读文件

我们采用开源工具JDOM来解析IOD和Petri网的xml文件。首先我们需要一个合适的解析器对象，由于本项目并不需要修改xml文件的树结构，为了可以占用更少内存并且获得更快的解析速度，我们采用SAX的方式解析本xml文件，所以本项目采用SAXBuilder而不是DOMBuilder作为解析器对象。

1. 实例化一个合适的解析器对象

SAXBuilder builder = new SAXBuilder();

1. 以包含XML数据的文件为参数，构建一个文档对象

Document doc = builder.build(new File(path));

1. 获到根元素

Element rootEl = doc.getRootElement();

一旦你获取了根元素，你就可以很方便地对它下面的子元素进行操作了，下面对Element对象的一些常用方法作一下简单说明：

|  |  |
| --- | --- |
| getChild("childname") | 返回指定名字的子节点,如果同一级有多个同名子节点，则只返回第一个；如果没有返回null值。 |
| getChildren("childname") | 返回指定名字的子节点List集合。这样你就可以遍历所有的同一级同名子节点。 |
| getAttributeValue("name") | 返回指定属性名字的值。如果没有该属性则返回null,有该属性但是值为空，则返回空字符串。 |
| getChildText("childname") | 返回指定子节点的内容文本值。 |
| getText() | 返回该元素的内容文本值。 |

表 3.1 jdom元素对象常用方法

## 3.2 IOD建模与解析

本小节讨论IOD的建模与解析，图3.1展示的是Visual Paradigm将图2.2 IOD图导出的xml文件的结构。我们根据这个文件分析xml文件的建模和解析方法。

整体来看，IOD根节点project下有三个子节点，分别是ProjectInfo，Models和Diagrams，ProjectInfo主要是一些项目信息，Diagrams描述的是如何绘图，这两个我们不用关心，本项目的重点在于关注如何对IOD图建模，也就是关注Models这个节点下的信息。



图 3.1 IOD xml文件结构示例

Models节点下主要有六类节点，分别是Frame， ModelRelationshipContainer, InitialNode, ActiityFinalNode, Interaction, DataType。为了解析出IOD的完整结构，需要对前五种节点进行分析和解析。

|  |  |  |
| --- | --- | --- |
| 节点名称 | | 节点意义 |
| Frame | | 时序图。IOD图中的每个时序图的内部信息，其中包含对象，生命线，激活等信息。 |
| Interaction | | 交互。IOD图中的时序图在整体中的信息，包含以其为终止的控制流和以其为出发的控制流。 |
| InitialNode | | 开始节点。同Interaction，只包含以其为出发的控制流 |
| ActivityFinalNode | | 终止节点。同Interaction，只包含以其为终止的控制流 |
| ModelRelationshipContainer | Message | 时序图中的消息。包括发出消息的生命线和接受消息的生命线的信息。 |
| ControlFlow | 控制流。包含发出控制流的交互和接受控制流的交互的信息。 |

表 3.2 IOD xml文件节点意义

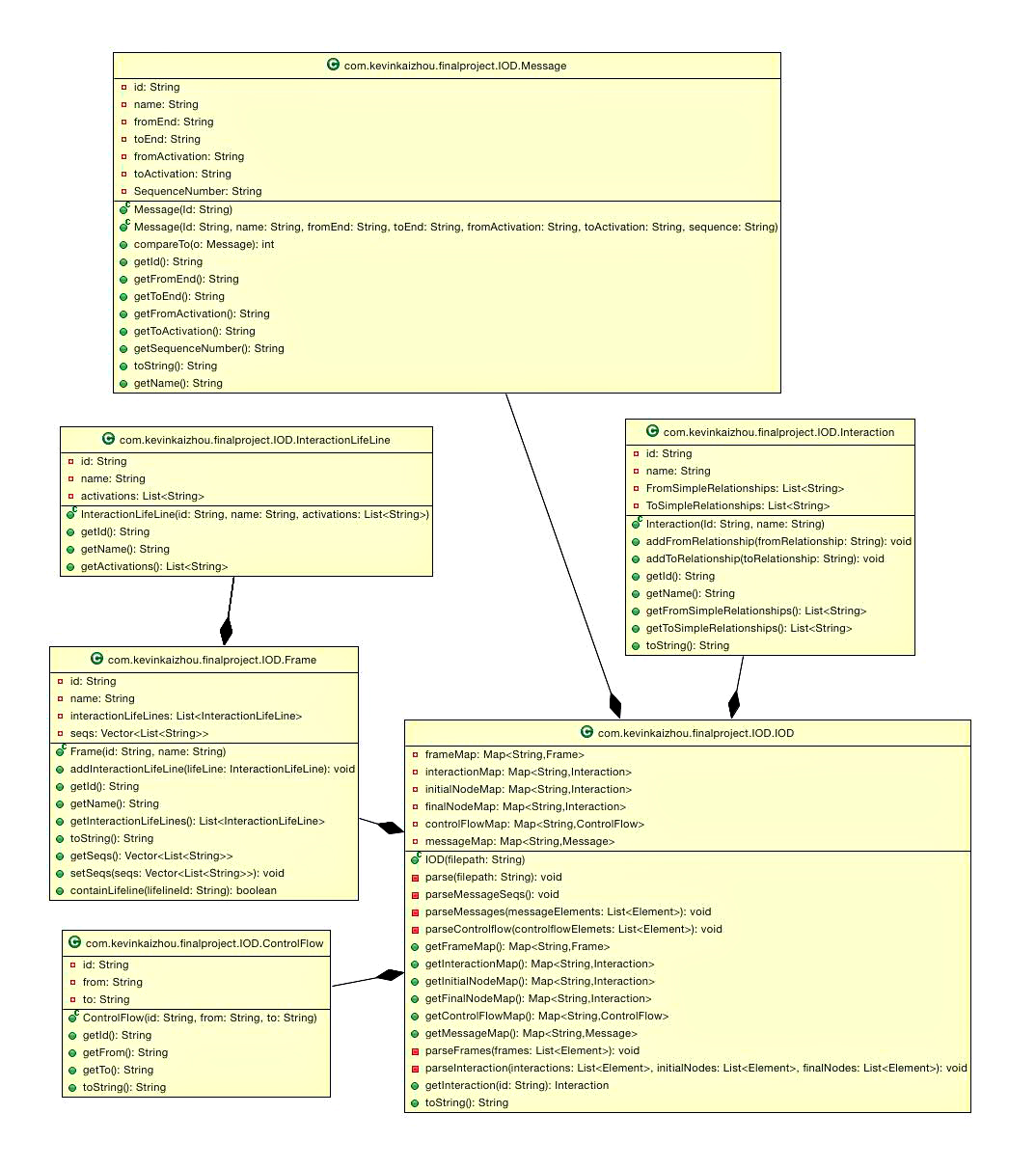


图 3.2 IOD模块类图

图3.2所示项目交互概览图（IOD）模块的类图，在IOD实例中保存了6个键值对。为了更好得与interaction对应上，frameMap用其名字作为键，其余Map均使用id作为键。

通过IOD类中一系列parse相关的方法，将xml文件读入内存。

## 3.3 Petri网建模与解析

Petri网建模相对于IOD建模要简单很多，图3.3所示一张由PIPE工具绘成的Petri网图，由PIPE生成的xml文件格式如图3.4所示。

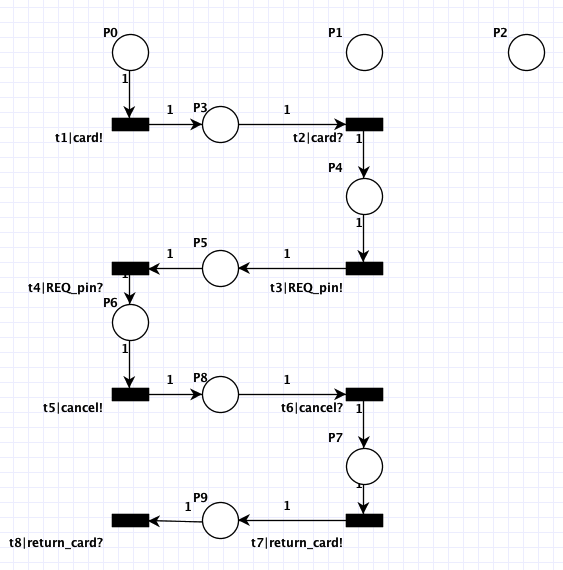


图 3.3 Petri网示例图



图 3.4 Petri网xml文件格式

在Petri网生成的xml文件中，主要包含place，transition和arc三类元素，表3.3对三类元素进行了分析和解析

|  |  |
| --- | --- |
| 节点名称 | 节点意义 |
| place | 库所。以id唯一标识 |
| transition | 变迁。因变迁的名字可能重复，所以在前面加上唯一标识符，并用竖线分割。 |
| arc | 有向弧。包含所有库所指向变迁和所有变迁指向库所的有向弧。以指向作为唯一标识。 |

表 3.3 Petri网xml文件节点意义

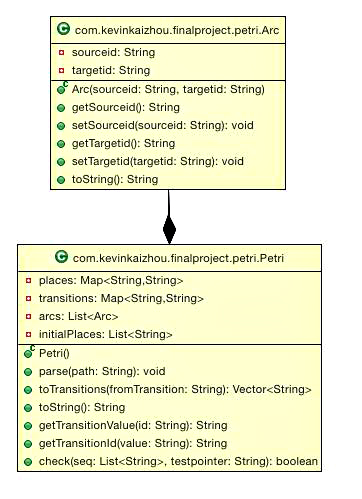


图 3.5 Petri网模块类图

图3.5是Petri网模块的类图，在Petri中，库所（place）和变迁（transition）均采用键值对方式存储。库所的健为其唯一标识Id，值为其名称，在本例中健值相同。变迁的健为其唯一标识Id，值为其名称（Id和名称用竖线隔开）。有一个列表存储所有有向弧，并且存储了所有初始的库所。

通过Petri类中得parse方法，将xml文件读入内存。

## 3.4 消息序列生成

### 3.4.1 消息序列生成需求获取

由于基于场景的并发模型存在一致性检验算法需要遍历IOD图中的时序图中得消息队列，所以需要记录每个时序图的所有可能的消息队列。在时序图中，消息的接受和发送我们称为事件，也就是2.1节所述的激活。事件的顺序（也就是消息发送和消息接受）是有时序图中对象的控制流可见的顺序和事件间的因果依赖推导得出的。为了不失一般性，在下面三种情况下我们认为事件e在e’之前：

1. 因果性：一个发送事件e和其相应地接受事件e’。比如说图2.1所示的时序图中，e5在e6之前。
2. 可控性：事件e和事件e’在一个对象上，并且e在e’之上，e’是发送事件。这个顺序说明了一个发送事件可以等待其他事件发生，另一方面可以说我们对接受消息顺序的可控性很低。比如说图2.1所示的时序图中，e6在e9之前。
3. 先进先出顺序：接受事件e在接受事件e’在同一个对象上，并且e在e’之上。相对应的发送事件e1和e1’在另一个对象上并且e1在e1’之上。

### 3.4.2 消息序列生成分析与设计

首先，需要能给所有的消息排序以确定哪个在上哪个在下，也就是要两个消息之间进行大小比较。分析Visual Paradigm生成的IOD图，可用消息的序列号进行排序，比如1.1在1.2之上，1.1在1.1.1之上。给Message类实现Comparable接口，并实现了compareTo方法：

|  |
| --- |
| //根据SequenceNumber比较两个message大小  @Override  **public** **int** compareTo(Message o) {  String[] s1 = SequenceNumber.split("\\.");  String[] s2 = o.getSequenceNumber().split("\\.");  **for** (**int** i = 0; i < Math.*min*(s1.length, s2.length); i++) {  **int** i1 = Integer.*parseInt*(s1[i]);  **int** i2 = Integer.*parseInt*(s2[i]);  **if** (i1 == i2) {  **continue**;  }  **return** i1 - i2;  }  **return** s1.length-s2.length;  } |

图 3.6 Message类compareTo方法

我们可以将消息序列生成问题简化为拓扑图排序问题，每一个事件可以作为拓扑图的一个节点，每个约束条件作为拓扑图的边，那么求消息队列的所有可能序列就转化为求拓扑图的所有可能排序。

为了说明方便，我们以图3.7所示的时序图为例，这张图包含了3.4.1小节所述的三种约束，加入约束条件后相应的拓扑图如3.8所示，先加入6个节点，然后根据三种不同类型约束依次加入：

1. 因果性约束：加入not\_approved!到not\_approved?，not\_possible!到not\_possible？和option!到option?三条约束。
2. 可控性约束：加入not\_approved?到not\_possible!，not\_approved?到option!和not\_possible!到option!三条约束。
3. 先进先出约束：加入not\_possible?到option?一条约束。

如图3.8所示，3.7时序图的所有可能的序列即为所有3.8可能的拓扑排序。

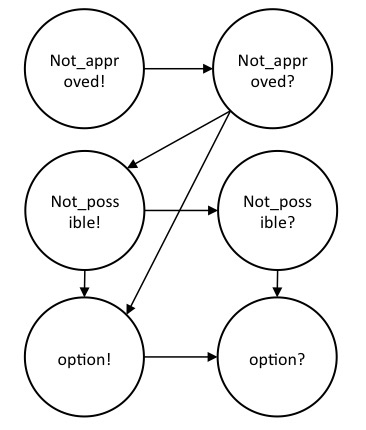
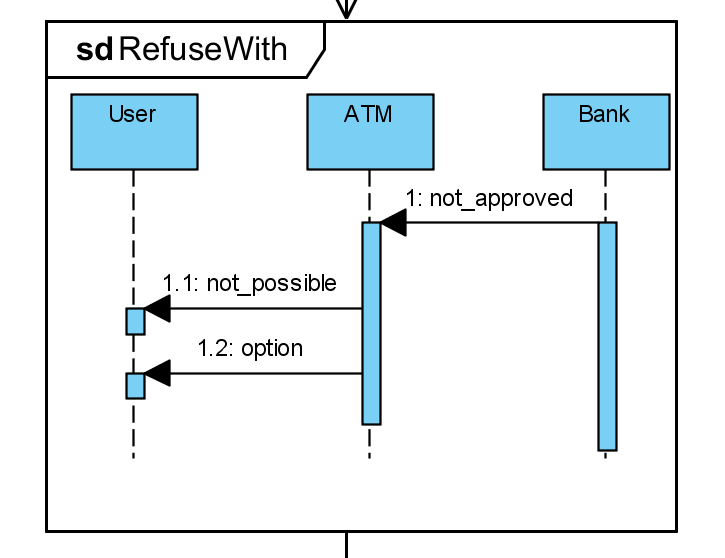


图 3.7 RefuseWith时序图 图 3.8 RefuseWith拓扑图

### 3.4.3 拓扑排序算法及相关描述

首先构建拓扑图的数据结构，在这个数据结构中保存着所有的边和节点。节点中保存着名称，出度和入度，出度通常指有向图中某点作为图中边的起点的次数之和，入度通常指有向图中某点作为图中边的终点的次数之和。在拓扑图中用键值对存储节点，健为其唯一标志名称。边包含了发出边的节点名和接受边的节点名，在拓扑图中用一个数组存储所有边。拓扑图数据结构包含添加节点，删除节点，添加边等方法，其中删除节点，会删除与这个节点有关的所有边。

根据3.4.2的方法将消息排序后，就可以确定RefuseWith中的约束关系，构建拓扑图。然后就根据拓扑排序算法将所有可能的排序算法保存在一个Vector中，具体算法见图3.9所示：

|  |
| --- |
| Vector<List<String>> messageSeqs;  Stack<String> tempSeq;    **private** **void** dfs(MessageGraph G) {  **if** (G.numberOfNodes() > 0 && G.nodesWithoutIn().size() == 0) {  System.***err***.println("拓扑图存在环，限制条件自相矛盾");  **return**;  }  **if** (G.numberOfNodes() == 0) {  recordStack();  }  **for** (String node : G.nodesWithoutIn()) {  Vector<Edge> vector = G.deleteNode(node);  tempSeq.push(node);  dfs(G);  tempSeq.pop();  G.addNode(node);  **for** (Edge edge : vector) {  G.addEdge(edge);  }  }  } |

图 3.9 拓扑排序递归算法

算法需要一个临时的栈来存放当前的路径，还需要一个Vector保存所有路径。算法的主要思想是深度优先遍历，搜寻拓扑图中入度为0的节点，将其加入栈中。将这个节点删除后继续搜索入度为0的节点，直到没有入度为0的节点。如果这个时候还有节点没有遍历到，说明拓扑图中存在环，也就是没有拓扑排序，如果所有节点都被遍历了，则将当前栈中保存的临时路径加入Vector中。依次返回，在搜寻其他入度为0的节点。

如图3.8所示的拓扑图，经过上述算法后，有两条拓扑路径：

1. not\_approved!, not\_approved?, not\_possible!, not\_possible?, option!, option?
2. not\_approved!, not\_approved?, not\_possible!, option!, not\_possible?, option?

也就是说，图3.7所示的时序图存在两条消息序列。

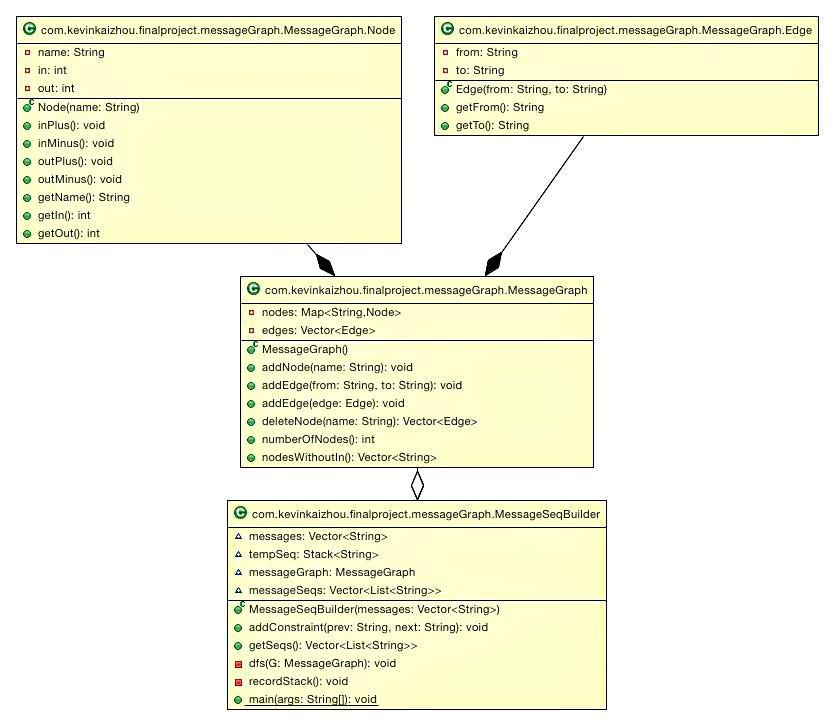


图 3.10 消息序列模块类图

# 第四章 验证算法核心模块的设计与实现

## 4.1 验证算法需求获取

验证算法模块是本工具的核心模块，本小节关注存在一致性检验，根据基于场景的规范检查Petri网。本论文中基于场景的规范就是包含IOD和时序图的UML交互模型，IOD是时序图的组成。存在一致性检验是检查一个给定IOD描述的场景是否一定会在Petri网中发生，或者是否一个IOD中禁止的场景一定不会在Petri网中发生。比如说，图4.1是两个可以描述铁路过路系统基于场景规范的时序图。左边的图是描绘火车过路前正常的场景，这个应该在图2.3所示的Petri网中出现。右边的图中Crossing\_secured这个消息在闸放下之前就发给了传感器，这个不会在图2.3所示Petri网中出现。

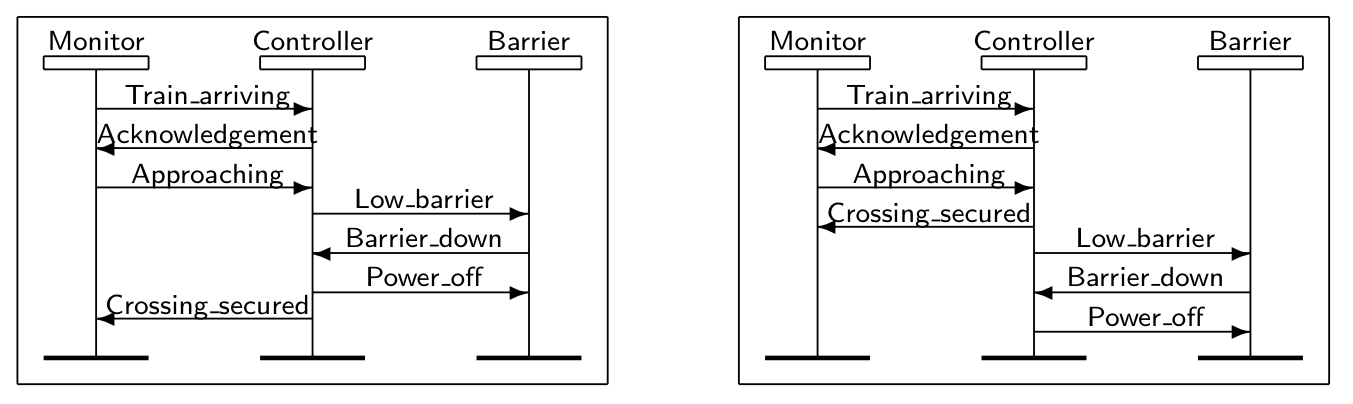


图 4.1 铁路过路系统存在一致性检验

验证算法要求能够在IOD图中找到一条路径在Petri网中发生，如果找到这样一条路，说明基于场景的并发模型存在一致性检验成功，如果不存在这样一条路，说明检验失败。

## 4.2 验证算法分析与设计

验证算法的核心采用深度优先遍历，搜索IOD图中的时序图，在时序图中选取一条消息序列在Petri网中看是否存在相应的路径，如果存在则递归下去遍历下一个节点，如果不存在则换一条消息序列试一下。如果此时序图中的所有消息序列都不能在Petri中找到相应的路径，则遍历前继节点的另一个后继节点，如果所有后继节点都不行，则说明不存在这样一条路径，存在一致性验证失败。如果遍历到IOD图的终止节点，就说明找到了一条IOD中的路径在Petri网中有相应的路径。

由于允许在IOD图中循环，为了避免在遍历时无限循环，在算法中需要做一个标记，如果走过一个时序图，标记走这个时序图在Petri网中相应的位置，如果再次走到这个时序图时还是在Petri网中这个位置，就避免再走这条路。

## 4.3 验证算法实现

|  |
| --- |
| **private** **boolean** check() {  **for** (String start : iod.getInitialNodeMap().keySet()) {  **if**(dfs(start,"")){  **return** **true**;  }  }  **return** **false**;  }    // 主要函数：深度优先遍历  **private** **boolean** dfs(String interactionId,String pipePointer) {  String interactionName = getVName(interactionId);  **if** (interactionName.equals("initial")) {  *tempPath*.add("initial");  **for** (String w : graph.adj(interactionId)) {  **if** (dfs(w, pipePointer)) {  **return** **true**;  }  }  **return** **false**;  }  **if** (interactionName.equals("final")) {  *tempPath*.add("final");  **return** **true**;  }  Frame frame = iod.getFrameMap().get(interactionName);  **for** (List<String> seq : frame.getSeqs()) {  String tid = petri.getTransitionId(seq.get(0));  //petri图中已走过，不会重新走  **if** (marked.containsKey(interactionId) && marked.get(interactionId).equals(tid)) {  **continue**;  }  marked.put(interactionId, tid);  *tempPath*.addAll(seq);  // check这个方法用来验证这个sequence在petri网中能否走通  **if** (petri.check(seq, pipePointer)) {  pipePointer = seq.get(seq.size()-1);  **for** (String w : graph.adj(interactionId)) {  **if** (dfs(w, pipePointer)) {  **return** **true**;  }  }  }  *tempPath*.removeAll(seq);  marked.remove(interactionId);  }  **return** **false**;  } |

图 4.2 存在一致性检验验证算法的实现

如图4.2所示为基于场景的并发模型存在一致性检验的核心算法，一下是对其的分析和解释：

首先是从IOD图的每个初始节点依次出发，只要找到这样的一条路径就停止遍历。

由于IOD图中的节点和时序图存储在不一样的地方，为了将两者关联起来，我们采用IOD图的名称作为唯一标识，可以根据名称找到IOD图中的节点对应的时序图。

如果是初始节点，则将自己加入路径，遍历他的所有后继节点。如果是终止节点，则说明找到了这条路，将自己加入路径后返回正确。

如果是普通节点，遍历其相应时序图的每一个消息序列，检查是否走过这个时序图和相应的Petri网，如果走过则检查下一跳消息序列，如果没走过则将这个对应加入纪录中。在Petri网中走过这个序列，如果检查成功，则继续深度遍历其后继节点。如果检查失败，检查下一条消息序列。

# 第五章 人机交互模块设计与实现

## 5.1 人机交互需求获取

基于场景的并发模型存在一致性检验工具的前端需求非常简单，用户选择IOD的xml文件并上传，选择Petri网的xml文件并上传，然后检查一致性，展示结果给用户。

在现代软件设计中，一个好的人交互体验至关重要。以下是本项目人机交互设计的几条标准。

1. 灵活性：界面可以随意拉伸以适应不同规格，不同分辨率的屏幕。
2. 可用性：由于本工具在处理大文件时可能需要耗时比较久，需加入控制台以告知用户现在进行到哪一步。
3. 安全性：避免用户做出错误操作，如遇到错误用合理的方式告知用户。

## 5.2 界面设计与实现

为适应灵活性需求，本项目采用java swing中的GridBagLayout作为工具界面的布局管理器，GridBagLayout是Swing中最灵活但也是最复杂的布局管理器。GridBagLayout从它的名字中你也可以猜到，它同GridLayout一样，在容器中以网格形式来管理组件。但GridBagLayout功能要来得强大得多。

1. GridBagLayout管理的所有行和列都可以是大小不同的。
2. GridLayout把每个组件限制到一个单元格，而GridBagLayout并不这样:组件在容器中可以占据任意大小的矩形区域，GridBagLayout通常由一个专用类来对他布局行为进行约束，该类叫GridBagConstraints。

经过设计，最后效果图如图5.1所示，可以随意拉伸，以适应不同屏幕尺寸和像素。

如图5.1所示，在界面设计中加入了控制台这个组件，可以提醒用户当前处理步骤。如果遇到错误也可以方便得查看在哪一步出现了问题，关于控制台功能的详细实现在5.3节中描述。

本项目的安全性体现在两个方面，一是避免用户犯错，比如用FileNameExtensionFilter禁止上传非xml文件，二是出错会进行提示而不是程序崩溃，比如如果用户没有选择文件会用JOptionPane.*showMessageDialog*进行提示。

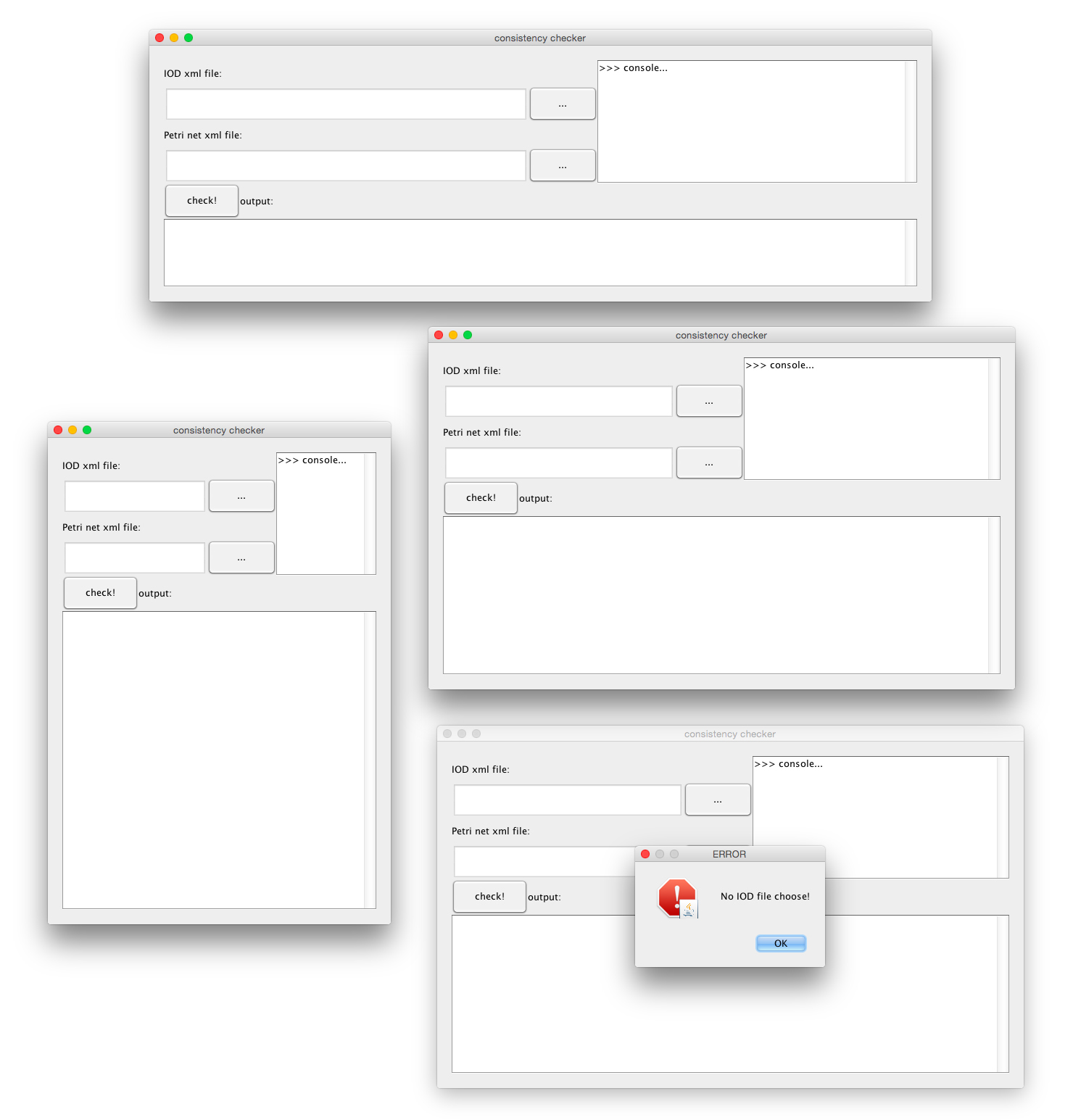


图5.1 界面设计效果图

## 5.3 处理信息实时展现设计与实现

控制台消息展现体现了本系统的MVC设计思想，下面对这一模块分析与展示：

为了实现能实时在界面显示当前处理进度的控制台功能，项目中使用了MessageShower这个接口，由界面实现它，控制器持有其引用，如图5.2所示，这符合MVC设计原则。

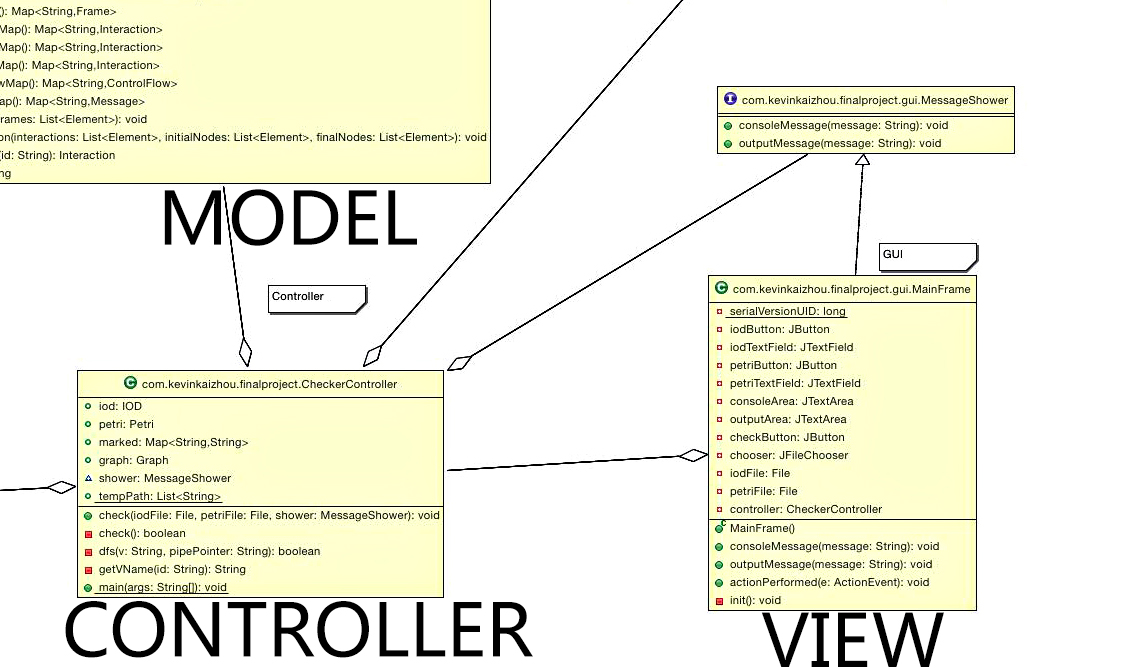


图 5.2 MVC部分类图

如图5.3所示，MessageShower中申明了两个方法，在view中实现了：这样当controller进行处理的时候就可以调用MessageShower的实例进行展示。

|  |
| --- |
| **public** **interface** MessageShower {  **public** **void** consoleMessage(String message);  **public** **void** outputMessage(String message);  }  **public** **class** MainFrame **extends** JFrame **implements** MessageShower,ActionListener{  ...  @Override  **public** **void** consoleMessage(String message) {  **if** (consoleArea!=**null**) {  consoleArea.append(">>> "+message+"\r\n");  consoleArea.setCaretPosition(consoleArea.getText().length());  }  }  @Override  **public** **void** outputMessage(String message) {  **if** (outputArea!=**null**) {  outputArea.append(message+"\r\n");  outputArea.setCaretPosition(outputArea.getText().length());  }  }  ...  } |

图5.3消息展示核心代码

# 第六章 整体概览与实例展示

## 6.1 整体概览

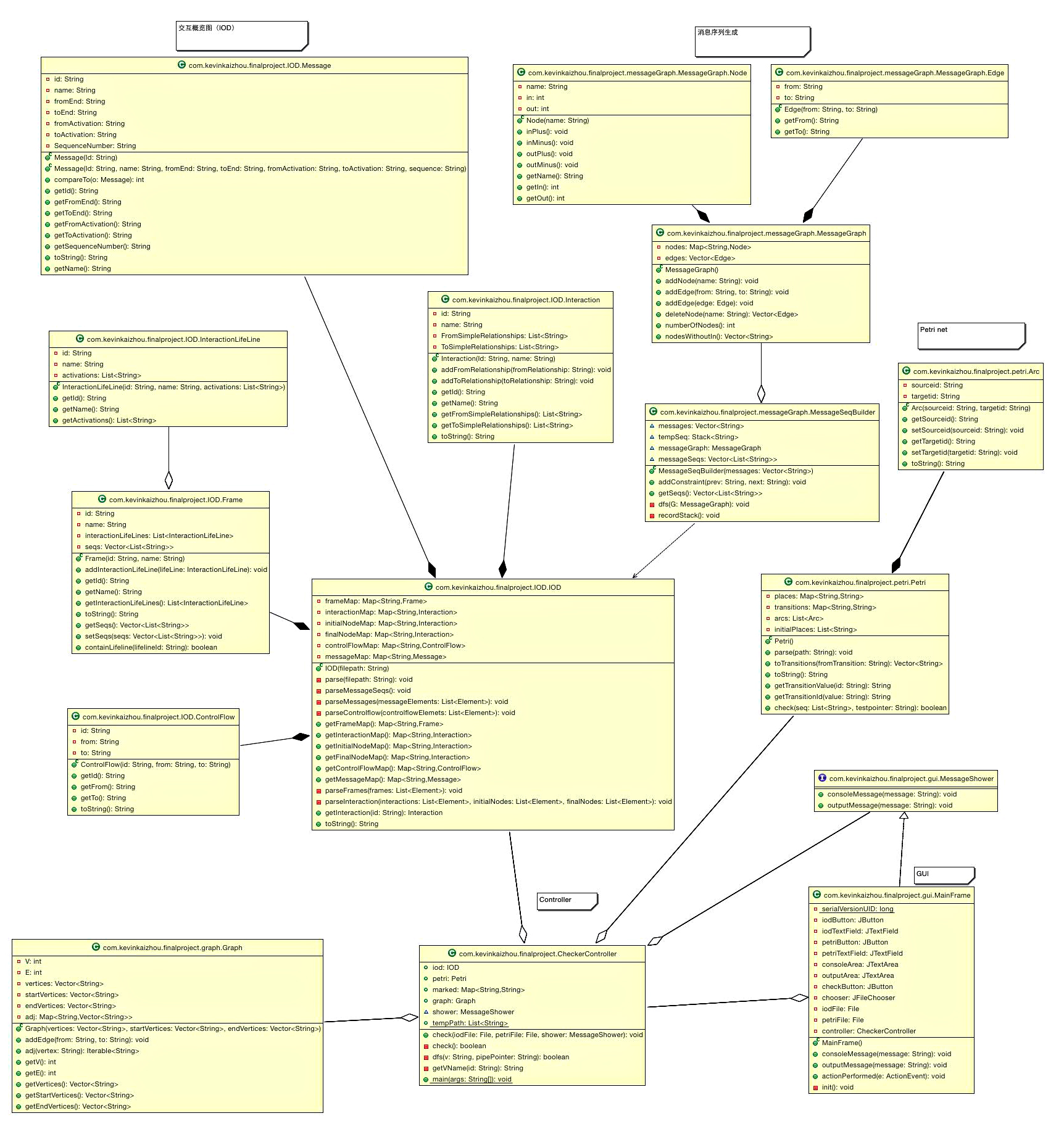
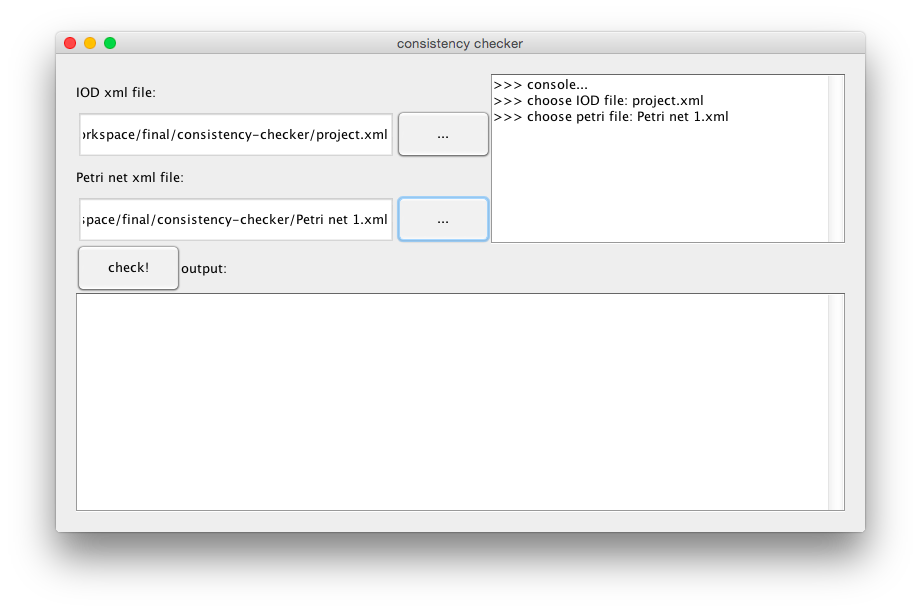


图6.1项目整体类图

图6.1所示是基于场景的并发模型存在一致性检验的整体类图展示，项目各模块之间划分得比较清晰，模块功能高内聚，并且用了设计模式等方法解耦合，保证了各模块的可扩展性。

## 6.2 实例展示



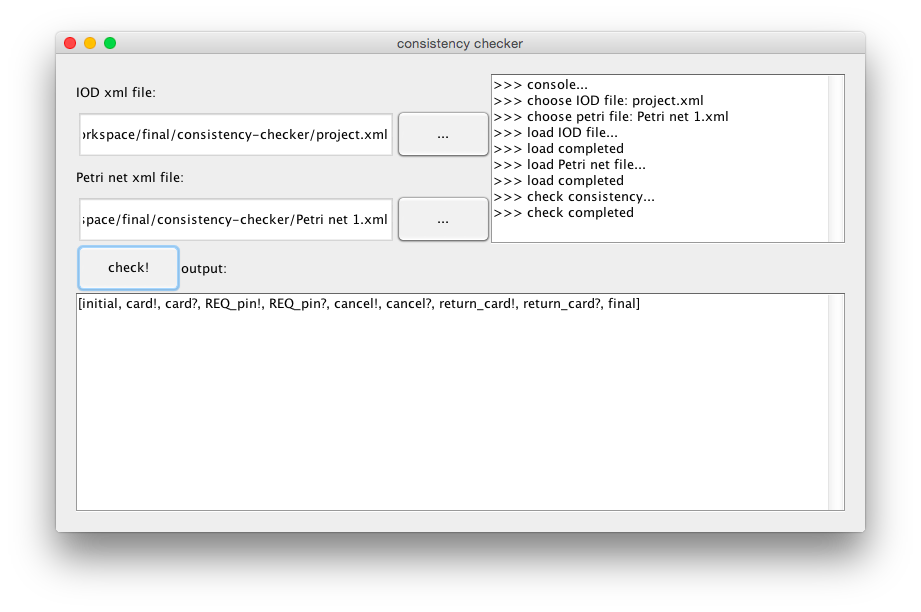


图 6.2 ATM系统部分存在一致性检验结果

下面用几个实例来展示本工具的效果：

首先是图2.2所示的IOD图和图3.3所示的Petri网的结果如图6.2，6.3所示。

下面是一个完整的ATM系统存在一致性检验工具的测试，图6.3为其IOD图，图6.4为其Petri网图。运行结果为：[initial, card!, card?, REQ\_pin!, REQ\_pin?, enter\_pin!, enter\_pin?, verify!, verify?, processing!, processing?, return\_card!, return\_card?, abort!, abort?, timeout! ,timeout? , card!, card?, REQ\_pin!, REQ\_pin?, enter\_pin!, enter\_pin?, verify!, verify?, processing!, processing?, invalid!, invalid?, INV\_MSG!, INV\_MSG?, take\_card!, take\_card?, final]，符合预期。说明此工具有效。

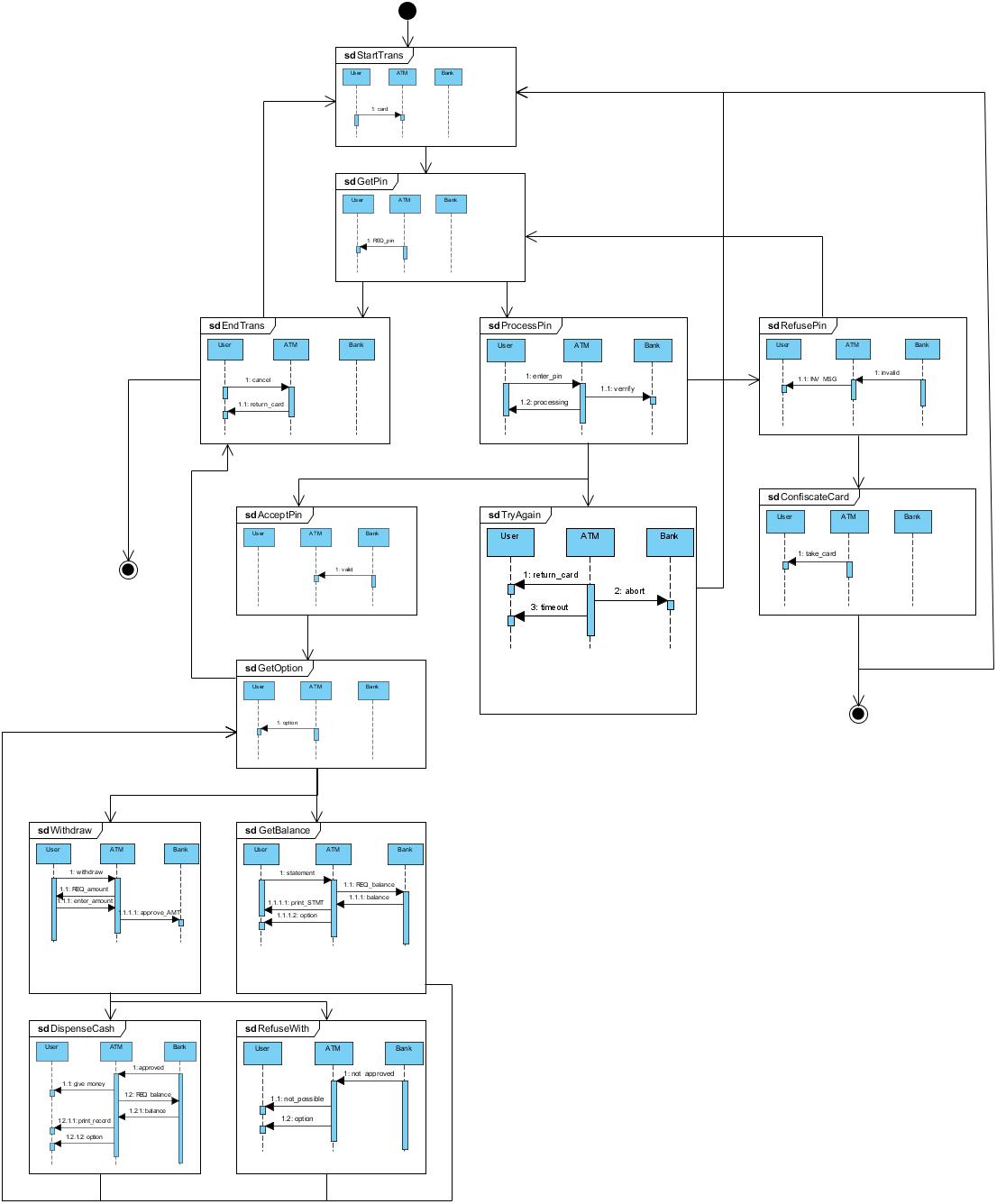


图 6.3 ATM IOD图

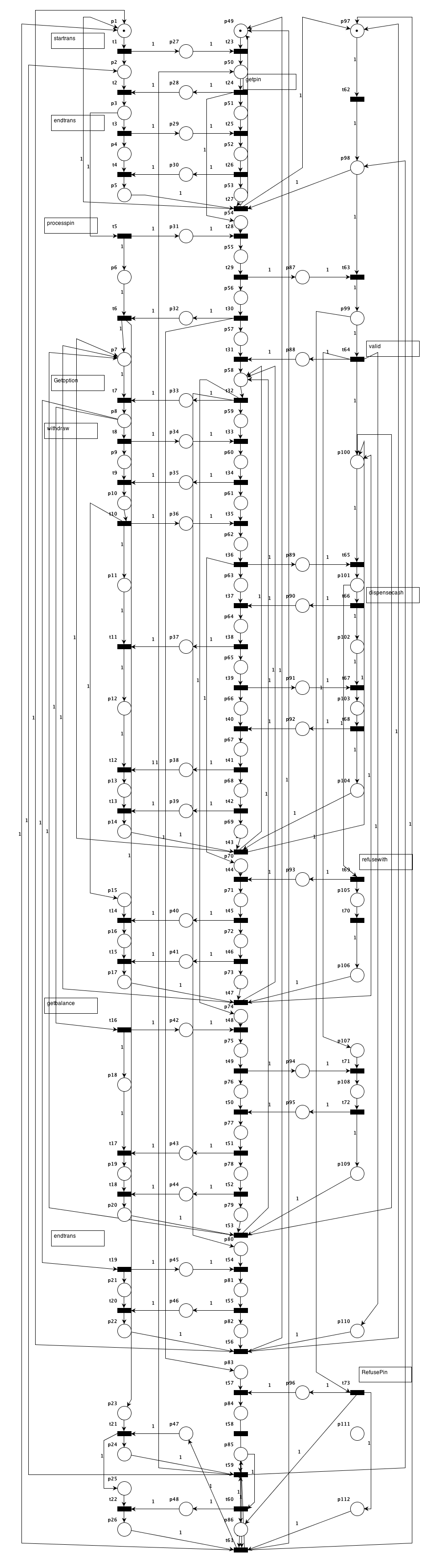


图 6.4 ATM Petri网

# 第七章 总结与展望

## 7.1 总结

本篇论文对基于场景的并发模型存在一致性检验工具做了总体的详细概述，重点描述了工具的设计与实现。首先介绍了相关概念，工具和技术，然后分别介绍了工具系统的三个模块。

1. 第一个模块是预处理模块，介绍了系统如何读文件和解析由Visual Paradigm生成的IOD的XML文件和由PIPE生成的Petri网的XML文件。并且重点介绍了基于拓扑排序的消息序列生成算法。
2. 第二个模块是验证算法核心模块，重点介绍了基于深度优先遍历的验证遍历算法。
3. 第三个模块是人机交互模块，介绍了界面的设计和实现，并重点介绍了基于MVC设计模式的控制台消息展示功能的设计与实现。

最后用两个实例展示说明了此工具的有效性。

## 7.2 不足和展望

基于场景的并发模型存在一致性检验工具为本人独立开发，由于水平有限，有一些不足和需要改进的地方。

没有进行XML的校验。虽然本工具只允许用户上传xml文件，但是没有对xml文件的内容进行校验，如果用户上传错误或者不符合规范的xml文件，可能会导致程序崩溃。在以后的开发中可以进行校验和提醒，防止出现这样的情况。

没有进行强制一致性检验。本工具只能进行了存在一致性检验的验证，下一步可以在工具中加入强制一致性检验的功能。

在人机交互上还不够友好。下一步可以考虑输出不止是一条路径而能在IOD图和Petri网中体现，就会更加直观和友好。

# 参考文献

[1] 胡军, 于笑丰, 张岩, 等. 基于场景构件式实时软件设计的一致性检验[J]. 软件学报, 2006, 17(1): 48-58.

[2] Robert Sedgewick，Kevin Wayne，谢路云，算法（第4版），人民邮电出版社，2012

[3] Class diagram - Wikipedia, the free encyclopedia：http://en.wikipedia.org/wiki/Class\_diagram

[4] Unified Modeling Language - Wikipedia, the free encyclopedia：http://en.wikipedia.org/wiki/Unified\_Modeling\_Language

[5] Sequence diagram - Wikipedia, the free encyclopedia：http://en.wikipedia.org/wiki/Sequence\_diagram

[6] Interaction overview diagram - Wikipedia, the free encyclopedia：http://en.wikipedia.org/wiki/Interaction\_overview\_diagram

[7] Petri net - Wikipedia, the free encyclopedia：http://en.wikipedia.org/wiki/Petri\_net

[8] 康保军. VP-UML 系统建模工具研究[J]. 软件工程师, 2014, 7: 019.

[9] List of Unified Modeling Language tools - Wikipedia, the free encyclopedia：http://en.wikipedia.org/wiki/List\_of\_Unified\_Modeling\_Language\_tools

[10] Platform Independent Petri net Editor 2：http://pipe2.sourceforge.net/

[11] Software Design Tools for Agile Teams, with UML, BPMN and More：http://www.visual-paradigm.com/

[12] 霍敏霞. 基于Petri网的并发程序测试路径生成[D]. 西南大学, 2011. DOI:10.7666/d.y1882620.

[13] JDOM\_百度百科：http://baike.baidu.com/link?url=KSuCWlVadOrIN2ziLzkw4YfjLfaeOuh-3jnn7fsJhbWGkETL8ukByJ7RidseJ3GwrMKTXbTsr4nBYFWMu4i34\_

[14] JDOM：http://www.jdom.org/

[15] Depth-first search - Wikipedia, the free encyclopedia：http://en.wikipedia.org/wiki/Depth-first\_search

[16] Topological sorting - Wikipedia, the free encyclopedia：http://en.wikipedia.org/wiki/Topological\_sorting

[17] Model–view–controller - Wikipedia, the free encyclopedia：http://en.wikipedia.org/wiki/Model-view-controller

[18] Swing (Java) - Wikipedia, the free encyclopedia：http://en.wikipedia.org/wiki/Swing\_(Java)

[19] 深入学习GridBagLayout：http://www.cnblogs.com/renyuan/archive/2012/10/15/2723809.html

# 致谢

大学四年的时间悄然逝去，转瞬已到尾声，经过几个月的艰苦奋斗，这篇毕业论文终于要接近尾声了。在项目和构建论文的过程中，我遇到了很多难题，也有很多迷茫的时刻，再次，我要向所有向我提供过指导、帮助和支持的人表示最真诚的感谢。

首先，我要感谢我的指导老师潘敏学老师。从毕业论文的选题，到项目的设计和实现，到论文提纲和内容的拟定，以及论文内容的修改和最后的定稿，潘老师一直都给予我细致的教诲和耐心的指导，特别是在我项目遇到瓶颈不知道怎么进行的时候，他耐心得一遍一遍给我解释算法，鼓励我不要放弃继续尝试，最后在潘老师的帮助下我攻克一个有一个难关，完成了项目。在此，我想要再次向潘老师表示我衷心的谢意。

其次，我要感谢学院各位领导和老师，学院为我们提供了良好的学习环境和设备支持，老师勤勤恳恳备课上课，让我在本科阶段打下了扎实的计算机基础，为我研究生生涯做好了铺垫。我还要感谢在软院一起学习，生活，做项目的同学们，感谢你们提供的无私帮助和照顾。特别感谢舍友尹心成，武慧凯，周海兵和姚远同学在本项目遇到算法瓶颈时帮我一起分析和给予宝贵的建议，没有你们的帮助，本项目很难完成。感谢王晨和宗岩琦同学在我遇到问题时困苦时的陪伴和开导。

由于我的学术水平有限，所写论文难免有不足之处，恳请各位老师和学友批评和指正。

****

****

**本 科 毕 业 设 计**

**院 系** 软件学院

**专 业** 软件工程

**题 目** 基于场景的并发模型存在一致性检验工具的设计与实现

**年 级** 2011级 **学 号** 111250250

**学生姓名** 周锴

**指导教师** 潘敏学 **职 称** 讲师

**论文提交日期**

**南京大学本科生毕业论文（设计）中文摘要**

毕业论文题目： 基于场景的并发模型存在一致性检验工具的设计与实现

软件学院 院系 软件工程 专业 2011级本科生姓名： 周锴

指导教师（姓名、职称）： 潘敏学 讲师

摘要：

如何有效地验证并发软件的设计是软件工程领域研究近些年的热点之一。场景可以描述具体的交互，所以被广泛地用于描述并发软件的设计，很容易为用户和领域专家使用。在本论文中，我们用统一建模语言（Unified Modeling Language 简称：UML）交互模型作为基于场景的规范，包含UML2.0的交互概览图（Interaction Overview Diagram 简称：IOD）和顺序图。Petri网可以很自然得用一种图形化的方式为并发约束建模，常被用来为系统规范和设计建模。通常，UML交互图被用来描述直接由用户提供的需求，而Petri网图被用来为由技术专家设计的工作流建模。所以，保持两个模型一致性即是保证了需求和设计的一致性，对保证软件质量是必须和重要的。本论文提出的基于场景的并发模型存在一致性检验工具，可以解决petri网建模的并发系统和UML交互模型表示的基于场景的规范的存在一致性检验问题。

为探究基于场景的并发模型存在一致性检验工具实现的解决方案，本文首先探讨了相关的概念，工具和技术。然后重点讨论了存在一致性算法的设计与实现，存在一致性算法需要在描述设计的Petri网中找到一条路径和描述需求的IOD图中的一条路径匹配，本算法采用深度优先遍历搜索IOD图，遍历的同时在Petri网中进行搜索和验证，最终找到在两张图中各自找到一条路径匹配则存在一致性检验成功，如果没找到则验证失败。最后通过两个实例验证了本工具的正确性和高效性。

关键词： 基于场景规范 存在一致性检验 深度优先遍历

**南京大学本科生毕业论文（设计）英文摘要**

THESIS： Design and Implementation of Scenario-Based Consistency Verification for Concurrent Models System

DEPARTMENT： Software Institute

SPECIALIZATION: Software Engineering

UNDERGRADUATE: Kai Zhou

MENTOR: Minxue Pan, Lecturer

ABSTRACT：

How to verify the design of concurrent software has become one of the focuses in software engineering research in these years. Scenarios are widely used as a required technique since they describe concrete interactions and are easy for customers and domain experts to use. In this paper, we use UML interaction models as scenario-based specifications, which consist of UML2.0 interaction overview diagrams (IODs) and sequence diagrams. Petri nets can model concurrency constraints in natural way, and are often used in modeling system specifications and designs. Usually, UML interaction models are used to describe the requirements provided directly by the customers, while Petri nets are used to model the workflow synthesized by the domain and technical experts. So it is necessary and important to keep the consistency between these two kinds of models for software quality assurance. In this paper, we use Petri nets to model concurrent systems, and consider the problem of checking Petri nets for the scenario-based specifications expressed by UML interaction models. The Scenario-Based Consistency Verification for Concurrent Models System is developed to solve the existential consistency checking problem.

To find a solution to solve this problem, this paper discussed related concepts, tools and techniques first. This tool is developed using Java programming language, MVC design pattern and cross-platform GUI maker Swing. We use Visual Paradigm to make UML interaction overview diagram (IOD) and export this as XML format. PIPE, another open-source tool, is used to build Petri net and export XML file as another input file. Our system can parse these XML file and read them into memory and then use Depth-First-Search algorithm to go through IOD, if we find a match path in Petri net, then the verification succeed. In the last, we use two examples to prove the correctness and efficiency of this system.

KEY WORDS: scenario-based specification, existential consistency checking problem, Depth-First-Search algorithm

**目 录**

2.3.4 MVC 1

2.3.5 java swing 1

2.3 本章小结 2

第三章 预处理模块的设计与实现 I

3.1 读文件 I

3.2 IOD建模与解析 I

3.3 Petri网建模与解析 IV

3.4 消息序列生成 VI

3.4.1 消息序列生成需求获取 VI

3.4.2 消息序列生成分析与设计 VII

3.4.3 拓扑排序算法及相关描述 VIII

第四章 验证算法核心模块的设计与实现 X

4.1 验证算法需求获取 X

4.2 验证算法分析与设计 X

4.3 验证算法实现 XI

第五章 人机交互模块设计与实现 XIII

5.1 人机交互需求获取 XIII

5.2 界面设计与实现 XIII

5.3 处理信息实时展现设计与实现 XIV

第六章 整体概览与实例展示 XVII

6.1 整体概览 XVII

6.2 实例展示 XVIII

第七章 总结与展望 XXI

7.1 总结 XXI

7.2 不足和展望 XXI

参考文献 XXII

图目录 III

表目录 IV

第一章 引言 1

1.1 研究背景和意义 1

1.2 相关工作 1

1.3 论文的主要工作和组织结构 1

第二章 工具和技术概述 3

2.1 概念 3

2.1.1 UML时序图 3

2.1.2 交互概览图（IOD） 4

2.1.3 Petri网 5

2.2 工具 8

2.2.1 Visual Paradigm 8

2.2.2 PIPE 8

2.3 技术 9

2.3.1 XML文件解析 9

2.3.2 深度优先遍历 10

2.3.3 拓扑排序 10

# 图目录

图 2.1 火车过路系统的时序图表示 1

图 2.2 交互概览图（IOD）示例 3

图 2.3 铁路过路系统Petri网模型 5

图 2.4 Visual Paradigm工具图标 6

图 2.5 PIPE2.0工具图标 6

图 2.6 深度优先搜索伪代码 8

图 2.7 MVC模式 9

图 3.1 IOD xml文件结构示例 12

图 3.2 IOD模块类图 13

图 3.3 Petri网示例图 14

图 3.4 Petri网xml文件格式 15

图 3.5 Petri网模块类图 16

图 3.6 Message类compareTo方法 17

图 3.9 拓扑排序递归算法 19

图 3.10 消息序列模块类图 19

图 4.1 铁路过路系统存在一致性检验 20

图 4.2 存在一致性检验验证算法的实现 22

图5.1 界面设计效果图 24

图 5.2 MVC部分类图 25

图5.3消息展示核心代码 26

图6.1项目整体类图 27

图 6.2 ATM系统部分存在一致性检验结果 28

图 6.3 ATM IOD图 29

图 6.4 ATM Petri网 30

# 

# 表目录

表 3.1 jdom元素对象常用方法 11

表 3.2 IOD xml文件节点意义 13

表 3.3 Petri网xml文件节点意义 15

# 第一章 引言

## 1.1 研究背景和意义

场景可以描述具体交互，并因此更容易被用户和领域专家使用。基于场景的规格，比如消息队列图，UML交互概览图，提供了一个描述系统需求的直观可见的方式。这些规格专注于并发和分布式系统中实体之间的信息交换。

统一建模语言（UML）在1997年成为对象管理组织（OMG）的一个标准，UML交互模型，如顺序图和活动图正变成软件开发过程的一个主要工具。作为活动图的一个变体，以顺序为元素的UML2.0交互概览图专注于流控制的概览和实体之间的信息交互。Petri网图是一个正式并且图形化的语言，很适合来为有并发的系统建模。我们通常在系统规格和设计时同时需要UML交互模型和Petri网图。

通常，UML交互图和Petri网图被用在不同的软件开发过程。即使用在同一个过程，比如软件需求分析过程，UML交互模型被通常用来描述直接由用户提供的基于场景的需求，而在设计阶段Petri图被用来为领域和技术专家合成的工作流建模。所以，为软件质量保证来保持两种模型的一致性是必要以及重要的。

保持两种模型一致性需要解决两个验证问题，存在一致性检验和强制一致性检验。存在性一致性检验是用来检查是否一个由给定IOD描述的场景一定会在Petri网图中发生。存在性一致性关乎到规格和验证的安全性。强制性一致性检验是用来检查一个强制性一致性规格在Petri中是否符合，意味着如果一个给定时序图描述的相关场景在Petri网图中出现，可以立即在IOD中找到对应的场景。

## 1.2 相关工作

对场景建模的方法主要分为用形式化规范语言描述以及用图形化的建模语言描述。形式化规范语言是在系统分析和设计中用一种比编程语言更高层次的视角来描述系统，常见的有CSP，

验证场景规约

## 1.3 论文的主要工作和组织结构

本文主要介绍了基于场景的并发模型存在一致性检验的相关理论知识，以及一个基于场景的并发模型存在一致性检验工具的设计和实现。主要是对基于场景的并发模型存在一致性检验的理论知识进行描述，同时描述一个基于场景的并发模型存在一致性检验工具的设计和实现过程以及这个过程中出现的一些问题。

第一章：概述和前言部分，主要介绍了项目背景和意义，并描述了该论文的主要工作。

第二章：主要介绍基于场景的并发模型存在一致性检验工具的开发和实现过程中用到的相关理论知识和理论研究，以及使用的工具和技术。

第三章：主要介绍了预处理模块的设计与实现，包括读文件，xml文件的解析，和IOD和Petri网的数据结构。着重介绍了基于拓扑排序的消息序列生成的算法。

第四章：主要介绍了验证算法核心模块的设计与实现。讨论了其需求的产生，进行了分析和设计，并展示了核心代码。

第五章：主要介绍了人机交互模块的设计与实现，着重介绍了利用MVC设计模式实现控制台实时展示的功能。

第六章：对项目进行了整体的展示，并用两个实例验证了工具的可用性。

第七章：总结该工具已实现的功能，探讨工具的缺点和不足，并指出该工具未来的扩展和发展方向。

# 第二章 工具和技术概述

## 2.1 概念

### 2.1.1 UML时序图

统一建模语言（Unified Modeling Language，简称：UML）是非专利的第三代建模和规约语言。UML是一种开放的方法，用于说明、可视化、构建和编写一个正在开发的、面向对象的、软件密集系统的制品的开放方法。UML展现了一系列最佳工程实践，这些最佳实践在对大规模，复杂系统进行建模方面，特别是在软件架构层次已经被验证有效。

UML集成了Booch，OMT和面向对象软件工程的概念，将这些方法融合为单一的，通用的，并且可以广泛使用的建模语言。UML打算成为可以对并发和分布式系统的标准建模语言。UML语言是一种能够支持模型化和图形化的语言，它能够为软件生产的各个阶段提供模块化和可视化的建模支持。

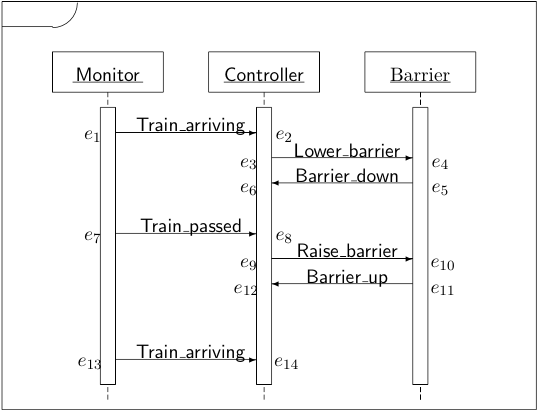


图 2.1 火车过路系统的时序图表示

时序图（Sequence Diagram），亦称为序列图或循序图，是一种UML行为图。它通过描述对象之间发送消息的时间顺序显示多个对象之间的动态协作。它可以表示用例的行为顺序，当执行一个用例行为时，时序图中的每条消息对应了一个类操作或状态机中引起转换的触发事件。时序图包含四类元素，他们分别是：

1. 对象 (Object)
2. 生命线 (Lifeline)
3. 消息 (Message)
4. 激活 (Activation)

在图2.1.1的火车过路系统时序图例子中，这个系统有一个火车通过检测仪（Monitor），一个控制器（Controller）控制火车十字系统的一个闸（Barrier）。当检测仪检测到火车正在靠近是，它发一个消息给控制器去降低闸。当火车离开路口后，这个检测仪发一个信息给控制器升起闸。在这个例子中，Monitor，Controller和Barrier是三个对象，其各有一条生命线，Train\_arriving，Lower\_barrier等即为对象间交换的消息，e1,e2表示激活，我们一般给消息加上感叹号表示发出消息的激活，加上问号表示接受消息的激活，比如e1可用Train\_arriving!表示，e2用Train\_arriving?表示。

### 2.1.2 交互概览图（IOD）

一个时序图只能描述一个场景，我们需要一个UML2.0的交互概览图（Interaction Overview Diagram）的简化版本来描绘多个场景。交互概览图是以时序图为节点的专注于描述控制流概览的模型。交互概览图定义了一个时序图的集合，描述了可能的循环和分支系统行为。

交互概要图是统一建模语言（UML）的一种图示，系在活动图的基础上，使用“交互框”作为元素，替代活动图中的“活动”。交互框间以“控制流”连接。交互概述图常用来描述用例的正常流与替代流之间的关系，作为内部协作图。

图2.2是一个简单地ATM机的交互概览图的一部分。在本论文中，交互概览图包含四类元素：

1. 交互（Interaction）
2. 控制流（Control Flow）
3. 开始节点（Initial Node）
4. 终止节点（End Node）

在本示例中有三个交互，一个开始节点，一个终止节点和五条控制流。

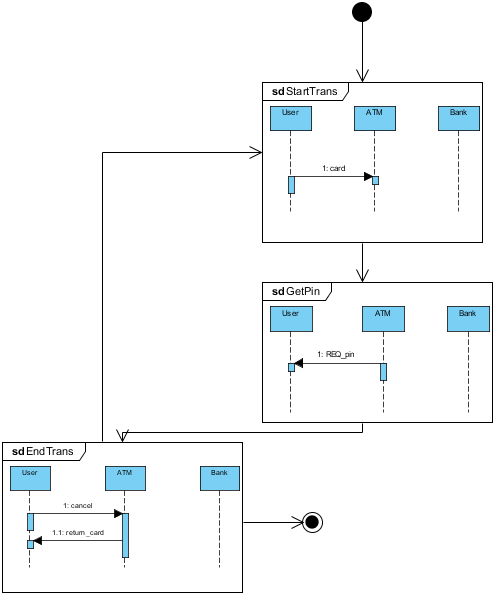


图 2.2 交互概览图（IOD）示例

### 2.1.3 Petri网

Petri网（Petri net）是对离散并行系统的数学表示。Petri网是1960年代由卡尔·亚当·佩特里发明的，适合于描述异步的、并发的计算机系统模型。Petri网既有严格的数学表述方式，也有直观的图形表达方式。由于Petri网能表达并发的事件，被认为是自动化理论的一种。研究领域趋向认为Petri网是所有流程定义语言之母。

经典的Petri网是简单的过程模型，由两种节点：库所和变迁，有向弧，以及令牌等元素组成的。Petri网的元素：

1. 库所（Place）圆形节点
2. 变迁（Transition）方形节点
3. 有向弧（Arc）是库所和变迁之间的有向弧
4. 令牌（Token）是库所中的动态对象，可以从一个库所移动到另一个库所。

Petri网的规则是：

1. 有向弧是有方向的
2. 两个库所或变迁之间不允许有弧
3. 库所可以拥有任意数量的令牌

如果一个变迁的每个输入库所（input place）都拥有数量足够的令牌时，该变迁即为被允许(enable)。一个变迁被允许时，变迁将发生(fire)，输入库所(input place)的令牌被消耗，同时为输出库所(output place)产生令牌。

图2.3表示的是一个火车过路系统的Petri网，在这张Petri网中有三个令牌从最上的库所发出。每个变迁代表一个消息的发送或者接受，如t1表示Train\_arriving!即为Train\_arriving这个消息的发送，t6表示Train\_arriving?即为Train\_arriving这个消息的接受。在这个系统中，当检测仪检测到火车靠近发送消息Train\_arriving到控制器，相应p1的令牌就到了p2和p7,控制器接受到Train\_arriving这个消息后，p7的令牌到了p13。控制器需要给检测仪发送一个确认信号Acknowledgement，即t7的Acknowledgement!，p13的令牌移动到p8和p14，由于p8和p2都有令牌，就可以接受消息Acknowledgement，即t2的Acknowledgement?。

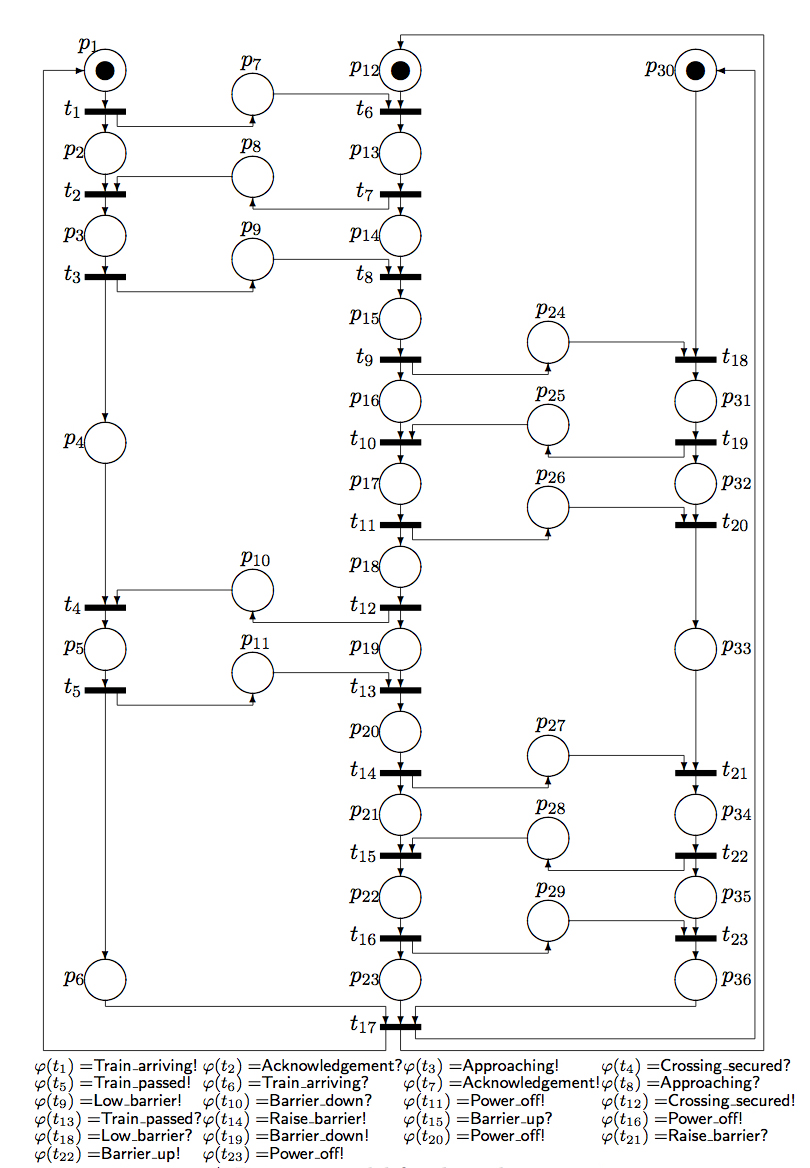


图 2.3 铁路过路系统Petri网模型

## 2.2 工具

### 2.2.1 Visual Paradigm



图 2.4 Visual Paradigm工具图标

随着面向对象技术的快速发展和应用，很多公司相继开发了支持面向对象技术的计算机辅助软件工程(Computer Aided Software Engineering，简称CASE)工具。在大浪淘沙的技术进步大潮中，有些工具渐渐退出了历史舞台，并逐渐被人们所遗忘；但有些不断发展，并为人们所接受，从而被广泛使用。Visual Paradigm for UML(简称VP-UML)是一款功能强大、跨平台、使用便捷、直观的UML系统建模和CASE工具，它可以整合在其他CASE工具或者其他IDE工具中，这种“一站购物式”的软件解决方案要远远优于常规的Model-Code-Deploy的开发进程。

对比市面上众多UML制图工具，Visual Paradigm因其友好的用户界面，便捷的xml导出功能等功能，被选为本项目的UML图制图工具。

### 2.2.2 PIPE



图 2.5 PIPE2.0工具图标

针对Petri网建模的工具有很多,较常用的有CNPTools、Visual Object Net++、 Tina、JFem、PIPE等。PIPE是Platfornl Independent Petri Net Editor的缩写,也是基于Java平台的。 具有图形编辑、托肯游戏动画、快速仿真、状态空间、凝聚状态空间、T-不变量、 P.不变量、结构分析、简单性能分析、文件格式转换、模块和文件格式深入分析等多项功能。下面是主要功能的简单介绍:图形编辑功能能够实现Petri网模型的建立、Petri网元素属性的编辑等;托肯游戏动画功能能够实现Petri网图形中库所元素间托肯的流动功能;快速仿真功能是指由PIPE建立的Petri网数学模型以及简化的计算方法得到的数学仿真;状态空间功能主要根据Petri网中安全性、有界性、死锁的相关概念来判断Petri网的相关性质。

对比市面上的Petri网建模工具，PIPE2.0因其友好的用户界面，跨平台的适用性被选为本项目的Petri网建模工具。

## 2.3 技术

### 2.3.1 XML文件解析

可扩展标记语言（eXtensible Markup Language，简称: XML）是一种标记语言。标记指计算机所能理解的信息符号，通过此种标记，计算机之间可以处理包含各种信息的文章等。如何定义这些标记，既可以选择国际通用的标记语言，比如HTML，也可以使用像XML这样由相关人士自由决定的标记语言，这就是语言的可扩展性。

现在对于XML文件解析工具主要有两种，一种是以DOM（Document Object Model）为基础的XML文件解析工具，它将XML文件整个读入内存，将文档一次性解析为树状结构，这种解析方式对内存占用比较大，构建一棵树比较缓慢，但可以随意存取文件树的任何部分，没有次数限制，还可以随意修改文件树，从而修改了XML文件。

另一种是以SAX（Simple API for XML）为基础的XML文件解析工具是一种基于事件的API，它并不在内存中生成文档树，而是在XML文档遍历的过程中采用一边对文档进行深度优先地遍历，一边在遍历的过程中对每个遇到的元素进行解析。SAX解析XML对内存占用比较小，但只能对文件按顺序剖析一遍，不支持对文件的随意存取。

本项目采用开源项目JDOM作为解析XML的工具，JDOM在2000年的春天被Brett McLaughlin和Jason Hunter开发出来，以弥补DOM及SAX在实际应用当中的不足之处。这些不足之处主要在于SAX没有文档修改、随机访问以及输出的功能，而对于DOM来说，JAVA程序员在使用时来用起来总觉得不太方便。在使用设计上尽可能地隐藏原来使用XML过程中的复杂性。利用JDOM处理XML文档将是一件轻松、简单的事。

### 2.3.2 深度优先遍历

深度优先搜索算法（Depth-First-Search），是搜索算法的一种。是沿着树的深度遍历树的节点，尽可能深的搜索树的分支。当节点v的所在边都己被探寻过，搜索将回溯到发现节点v的那条边的起始节点。这一过程一直进行到已发现从源节点可达的所有节点为止。如果还存在未被发现的节点，则选择其中一个作为源节点并重复以上过程，整个进程反复进行直到所有节点都被访问为止。

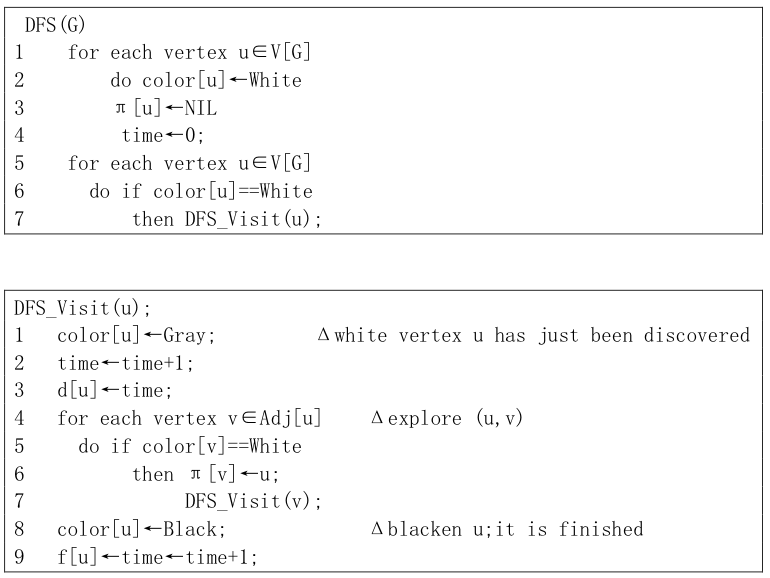


图 2.6 深度优先搜索伪代码

本项目利用深度优先遍历算法找到IOD中和Petri网匹配的一条路径。

### 2.3.3 拓扑排序

在图论中，由一个有向无环图的顶点组成的序列，当且仅当满足下列条件时，称为该图的一个拓扑排序（英语：Topological sorting）。

1. 每个顶点出现且只出现一次；
2. 若A在序列中排在B的前面，则在图中不存在从B到A的路径。

也可以定义为：拓扑排序是对有向无环图的顶点的一种排序，它使得如果存在一条从顶点A到顶点B的路径，那么在排序中B出现在A的后面。

在本项目中，我改进了拓扑排序的Kahn算法，解决了记录时序图中所有可能消息序列的问题。

# 第二章 工具和技术概述

## 2.1 概念

### 2.1.1 UML时序图

统一建模语言（Unified Modeling Language，简称：UML）是非专利的第三代建模和规约语言。UML是一种开放的方法，用于说明、可视化、构建和编写一个正在开发的、面向对象的、软件密集系统的制品的开放方法。UML展现了一系列最佳工程实践，这些最佳实践在对大规模，复杂系统进行建模方面，特别是在软件架构层次已经被验证有效。

UML集成了Booch，OMT和面向对象软件工程的概念，将这些方法融合为单一的，通用的，并且可以广泛使用的建模语言。UML打算成为可以对并发和分布式系统的标准建模语言。UML语言是一种能够支持模型化和图形化的语言，它能够为软件生产的各个阶段提供模块化和可视化的建模支持。

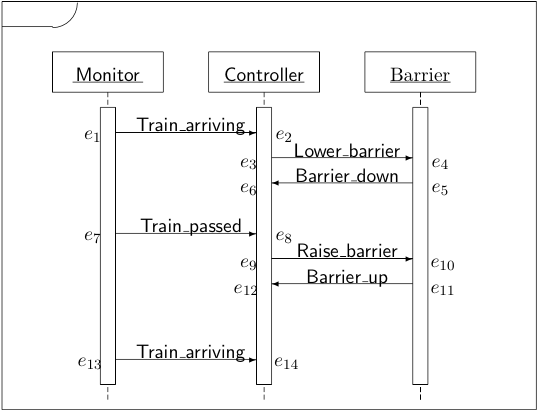


图 2.1 火车过路系统的时序图表示

时序图（Sequence Diagram），亦称为序列图或循序图，是一种UML行为图。它通过描述对象之间发送消息的时间顺序显示多个对象之间的动态协作。它可以表示用例的行为顺序，当执行一个用例行为时，时序图中的每条消息对应了一个类操作或状态机中引起转换的触发事件。时序图包含四类元素，他们分别是：

1. 对象 (Object)
2. 生命线 (Lifeline)
3. 消息 (Message)
4. 激活 (Activation)

在图2.1.1的火车过路系统时序图例子中，这个系统有一个火车通过检测仪（Monitor），一个控制器（Controller）控制火车十字系统的一个闸（Barrier）。当检测仪检测到火车正在靠近是，它发一个消息给控制器去降低闸。当火车离开路口后，这个检测仪发一个信息给控制器升起闸。在这个例子中，Monitor，Controller和Barrier是三个对象，其各有一条生命线，Train\_arriving，Lower\_barrier等即为对象间交换的消息，e1,e2表示激活，我们一般给消息加上感叹号表示发出消息的激活，加上问号表示接受消息的激活，比如e1可用Train\_arriving!表示，e2用Train\_arriving?表示。

### 2.1.2 交互概览图（IOD）

一个时序图只能描述一个场景，我们需要一个UML2.0的交互概览图（Interaction Overview Diagram）的简化版本来描绘多个场景。交互概览图是以时序图为节点的专注于描述控制流概览的模型。交互概览图定义了一个时序图的集合，描述了可能的循环和分支系统行为。

交互概要图是统一建模语言（UML）的一种图示，系在活动图的基础上，使用“交互框”作为元素，替代活动图中的“活动”。交互框间以“控制流”连接。交互概述图常用来描述用例的正常流与替代流之间的关系，作为内部协作图。

图2.2是一个简单地ATM机的交互概览图的一部分。在本论文中，交互概览图包含四类元素：

1. 交互（Interaction）
2. 控制流（Control Flow）
3. 开始节点（Initial Node）
4. 终止节点（End Node）

在本示例中有三个交互，一个开始节点，一个终止节点和五条控制流。

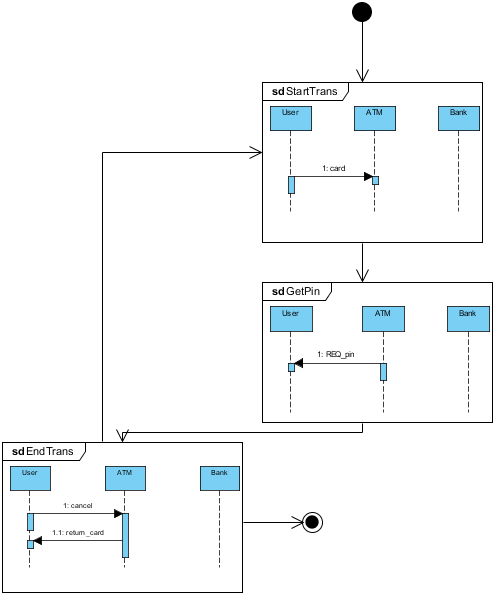


图 2.2 交互概览图（IOD）示例

### 2.1.3 Petri网

Petri网（Petri net）是对离散并行系统的数学表示。Petri网是1960年代由卡尔·亚当·佩特里发明的，适合于描述异步的、并发的计算机系统模型。Petri网既有严格的数学表述方式，也有直观的图形表达方式。由于Petri网能表达并发的事件，被认为是自动化理论的一种。研究领域趋向认为Petri网是所有流程定义语言之母。

经典的Petri网是简单的过程模型，由两种节点：库所和变迁，有向弧，以及令牌等元素组成的。Petri网的元素：

1. 库所（Place）圆形节点
2. 变迁（Transition）方形节点
3. 有向弧（Arc）是库所和变迁之间的有向弧
4. 令牌（Token）是库所中的动态对象，可以从一个库所移动到另一个库所。

Petri网的规则是：

1. 有向弧是有方向的
2. 两个库所或变迁之间不允许有弧
3. 库所可以拥有任意数量的令牌

如果一个变迁的每个输入库所（input place）都拥有数量足够的令牌时，该变迁即为被允许(enable)。一个变迁被允许时，变迁将发生(fire)，输入库所(input place)的令牌被消耗，同时为输出库所(output place)产生令牌。

图2.3表示的是一个火车过路系统的Petri网，在这张Petri网中有三个令牌从最上的库所发出。每个变迁代表一个消息的发送或者接受，如t1表示Train\_arriving!即为Train\_arriving这个消息的发送，t6表示Train\_arriving?即为Train\_arriving这个消息的接受。在这个系统中，当检测仪检测到火车靠近发送消息Train\_arriving到控制器，相应p1的令牌就到了p2和p7,控制器接受到Train\_arriving这个消息后，p7的令牌到了p13。控制器需要给检测仪发送一个确认信号Acknowledgement，即t7的Acknowledgement!，p13的令牌移动到p8和p14，由于p8和p2都有令牌，就可以接受消息Acknowledgement，即t2的Acknowledgement?。

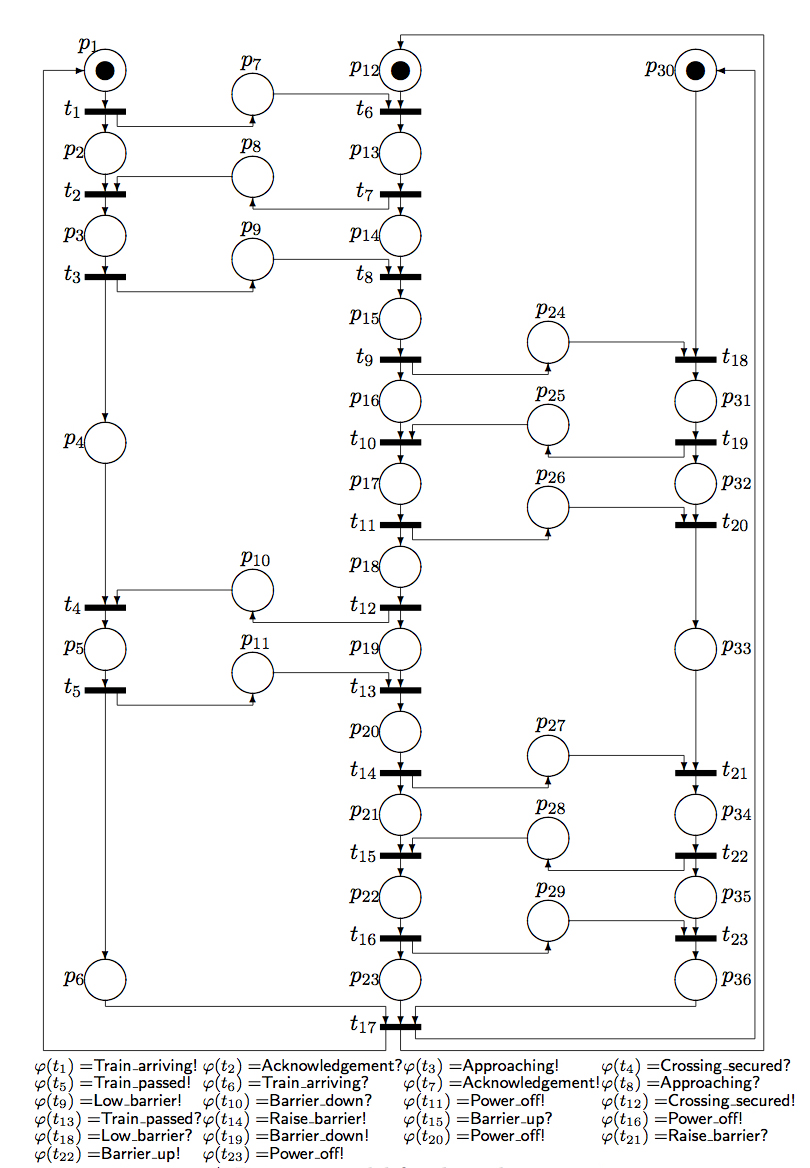


图 2.3 铁路过路系统Petri网模型

## 2.2 工具

### 2.2.1 Visual Paradigm



图 2.4 Visual Paradigm工具图标

随着面向对象技术的快速发展和应用，很多公司相继开发了支持面向对象技术的计算机辅助软件工程(Computer Aided Software Engineering，简称CASE)工具。在大浪淘沙的技术进步大潮中，有些工具渐渐退出了历史舞台，并逐渐被人们所遗忘；但有些不断发展，并为人们所接受，从而被广泛使用。Visual Paradigm for UML(简称VP-UML)是一款功能强大、跨平台、使用便捷、直观的UML系统建模和CASE工具，它可以整合在其他CASE工具或者其他IDE工具中，这种“一站购物式”的软件解决方案要远远优于常规的Model-Code-Deploy的开发进程。

对比市面上众多UML制图工具，Visual Paradigm因其友好的用户界面，便捷的xml导出功能等功能，被选为本项目的UML图制图工具。

### 2.2.2 PIPE



图 2.5 PIPE2.0工具图标

针对Petri网建模的工具有很多,较常用的有CNPTools、Visual Object Net++、 Tina、JFem、PIPE等。PIPE是Platfornl Independent Petri Net Editor的缩写,也是基于Java平台的。 具有图形编辑、托肯游戏动画、快速仿真、状态空间、凝聚状态空间、T-不变量、 P.不变量、结构分析、简单性能分析、文件格式转换、模块和文件格式深入分析等多项功能。下面是主要功能的简单介绍:图形编辑功能能够实现Petri网模型的建立、Petri网元素属性的编辑等;托肯游戏动画功能能够实现Petri网图形中库所元素间托肯的流动功能;快速仿真功能是指由PIPE建立的Petri网数学模型以及简化的计算方法得到的数学仿真;状态空间功能主要根据Petri网中安全性、有界性、死锁的相关概念来判断Petri网的相关性质。

对比市面上的Petri网建模工具，PIPE2.0因其友好的用户界面，跨平台的适用性被选为本项目的Petri网建模工具。

## 2.3 技术

### 2.3.1 XML文件解析

可扩展标记语言（eXtensible Markup Language，简称: XML）是一种标记语言。标记指计算机所能理解的信息符号，通过此种标记，计算机之间可以处理包含各种信息的文章等。如何定义这些标记，既可以选择国际通用的标记语言，比如HTML，也可以使用像XML这样由相关人士自由决定的标记语言，这就是语言的可扩展性。

现在对于XML文件解析工具主要有两种，一种是以DOM（Document Object Model）为基础的XML文件解析工具，它将XML文件整个读入内存，将文档一次性解析为树状结构，这种解析方式对内存占用比较大，构建一棵树比较缓慢，但可以随意存取文件树的任何部分，没有次数限制，还可以随意修改文件树，从而修改了XML文件。

另一种是以SAX（Simple API for XML）为基础的XML文件解析工具是一种基于事件的API，它并不在内存中生成文档树，而是在XML文档遍历的过程中采用一边对文档进行深度优先地遍历，一边在遍历的过程中对每个遇到的元素进行解析。SAX解析XML对内存占用比较小，但只能对文件按顺序剖析一遍，不支持对文件的随意存取。

本项目采用开源项目JDOM作为解析XML的工具，JDOM在2000年的春天被Brett McLaughlin和Jason Hunter开发出来，以弥补DOM及SAX在实际应用当中的不足之处。这些不足之处主要在于SAX没有文档修改、随机访问以及输出的功能，而对于DOM来说，JAVA程序员在使用时来用起来总觉得不太方便。在使用设计上尽可能地隐藏原来使用XML过程中的复杂性。利用JDOM处理XML文档将是一件轻松、简单的事。

### 2.3.2 深度优先遍历

深度优先搜索算法（Depth-First-Search），是搜索算法的一种。是沿着树的深度遍历树的节点，尽可能深的搜索树的分支。当节点v的所在边都己被探寻过，搜索将回溯到发现节点v的那条边的起始节点。这一过程一直进行到已发现从源节点可达的所有节点为止。如果还存在未被发现的节点，则选择其中一个作为源节点并重复以上过程，整个进程反复进行直到所有节点都被访问为止。

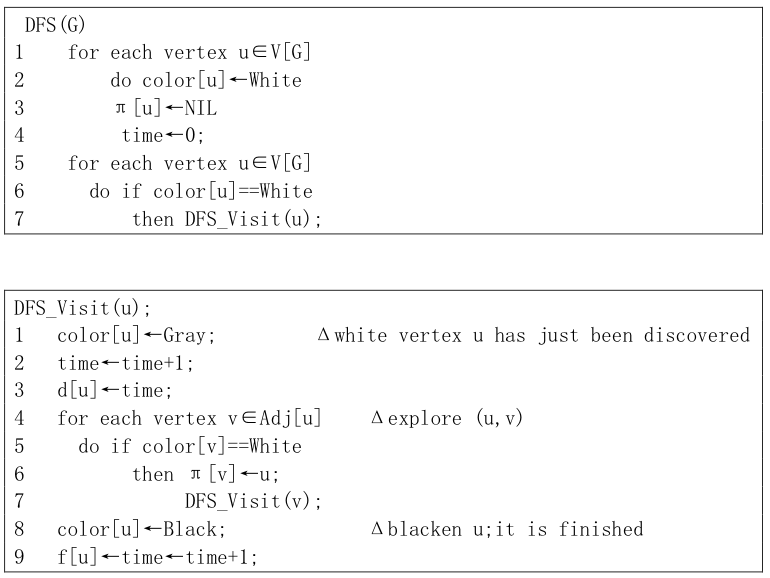


图 2.6 深度优先搜索伪代码

本项目利用深度优先遍历算法找到IOD中和Petri网匹配的一条路径。

### 2.3.3 拓扑排序

在图论中，由一个有向无环图的顶点组成的序列，当且仅当满足下列条件时，称为该图的一个拓扑排序（英语：Topological sorting）。

1. 每个顶点出现且只出现一次；
2. 若A在序列中排在B的前面，则在图中不存在从B到A的路径。

也可以定义为：拓扑排序是对有向无环图的顶点的一种排序，它使得如果存在一条从顶点A到顶点B的路径，那么在排序中B出现在A的后面。

在本项目中，我改进了拓扑排序的Kahn算法，解决了记录时序图中所有可能消息序列的问题。