状态图自动化建模和模型验证项目（M&MC）

状态图图形化模块的设计与实现

吴继鹏

南京大学软件学院
规格技术研究小组

Final Presentation, 2015

Outline

这篇报告包括：

- 支持 M&MC 项目的形式化规格方法以及相关理论背景
- M&MC 的项目背景、需求分析和概要设计
- 对基于状态机模型的形式化类规格进行可视化建模的详细设计与实现

这篇报告包括：

- 支持 M&MC 项目的形式化规格方法以及相关理论背景
- M&MC 的项目背景、需求分析和概要设计
- 对基于状态机模型的形式化类规格进行可视化建模的详细设计与实现

这篇报告包括：

- 支持 M&MC 项目的形式化规格方法以及相关理论背景
- M&MC 的项目背景、需求分析和概要设计
- 对基于状态机模型的形式化类规格进行可视化建模的详细设计与实现

这篇报告包括：

- 支持 M&MC 项目的形式化规格方法以及相关理论背景
- M&MC 的项目背景、需求分析和概要设计
- 对基于状态机模型的形式化类规格进行可视化建模的详细设计与实现

Outline

形式化规格的类别

- **固定原则** (fixed discipline), s := s i
- 任意原则 (arbitrary discipline), insert(s,i)
- 状态机模型
- 公理化表示
- 代数定义，少量公理 + 功能定义

形式化规格的类别

- **固定原则** (fixed discipline), s := s i
- **任意原则** (arbitrary discipline), insert(s,i)
- 状态机模型
- 公理化表示
- 代数定义，少量公理 + 功能定义

形式化规格的类别

- 固定原则 (fixed discipline), s := s i
- 任意原则 (arbitrary discipline), insert(s,i)
- 状态机模型
- 公理化表示
- 代数定义，少量公理 + 功能定义

形式化规格的类别

- 固定原则 (fixed discipline), s := s i
- 任意原则 (arbitrary discipline), insert(s,i)
- 状态机模型
- 公理化表示
- 代数定义，少量公理 + 功能定义

形式化规格的类别

- 固定原则 (fixed discipline), s := s i
- 任意原则 (arbitrary discipline), insert(s,i)
- 状态机模型
- 公理化表示
- 代数定义，少量公理 + 功能定义

形式化规格的类别

- 固定原则 (fixed discipline), s := s i
- 任意原则 (arbitrary discipline), insert(s,i)
- 状态机模型
- 公理化表示
- 代数定义，少量公理 + 功能定义

形式化规格的质量属性

- 形式化
- 可构建性
- 最小化
- 应用广度
- 可拓展性

形式化规格的质量属性

- 形式化
- 可构建性
- 最小化
- 应用广度
- 可拓展性

形式化规格的质量属性

- 形式化
- 可构建性
- 最小化
- 应用广度
- 可拓展性

形式化规格的质量属性

- 形式化
- 可构建性
- 最小化
- 应用广度
- 可拓展性

形式化规格的质量属性

- 形式化
- 可构建性
- 最小化
- 应用广度
- 可拓展性

形式化规格的质量属性

- 形式化
- 可构建性
- 最小化
- 应用广度
- 可拓展性

其他相关研究

- inspector, mutator（需要注释的方法）
- DBC（依赖于实现）
- Statecharts（Harel）

其他相关研究

- inspector, mutator（需要注释的方法）
- DBC（依赖于实现）
- Statecharts（Harel）

其他相关研究

- inspector, mutator（需要注释的方法）
- DBC（依赖于实现）
- Statecharts（Harel）

其他相关研究

- inspector, mutator（需要注释的方法）
- DBC（依赖于实现）
- Statecharts（Harel）

Outline

M&MC 中类的状态图模型定义

- $C = (S, S_0, T)$
- $S = s_1, s_2, \ldots, s_n (n\ 0)$
- $s_i = R_i(v_1), R_i(v_2), \ldots, R_i(v_k)$ 这已经是抽象化的状态了，一个状态可以包含若干种绝对状态，即由状态变量取值决定的状态。
- $T = t_1, t_2, \ldots, t_m (m\ 0)$，其中 t_i 是一个 S 到 S 的映射函数。一个 $transition t_i$ 被规格化为 (spre, spost, M, C)。

M&MC 中类的状态图模型定义

- $C = (S, S_0, T)$
- $S = s_1, s_2, \ldots, s_n (n\ 0)$
- $s_i = R_i(v_1), R_i(v_2), \ldots, R_i(v_k)$ 这已经是抽象化的状态了，一个状态可以包含若干种绝对状态，即由状态变量取值决定的状态。
- $T = t_1, t_2, \ldots, t_m (m\ 0)$，其中 t_i 是一个 S 到 S 的映射函数。一个 $transition t_i$ 被规格化为 (spre, spost, M, C)。

M&MC 中类的状态图模型定义

- $C = (S, S_0, T)$
- $S = s_1, s_2, \ldots, s_n (n\ 0)$
- $s_i = R_i(v_1), R_i(v_2), \ldots, R_i(v_k)$ 这已经是抽象化的状态了，一个状态可以包含若干种绝对状态，即由状态变量取值决定的状态。
- $T = t_1, t_2, \ldots, t_m (m\ 0)$，其中 t_i 是一个 S 到 S 的映射函数。一个 $transition t_i$ 被规格化为 (spre, spost, M, C)。

M&MC 中类的状态图模型定义

- $C = (S, S_0, T)$
- $S = s_1, s_2, \ldots, s_n (n\ 0)$
- $s_i = R_i(v_1), R_i(v_2), \ldots, R_i(v_k)$ 这已经是抽象化的状态了，一个状态可以包含若干种绝对状态，即由状态变量取值决定的状态。
- $T = t_1, t_2, \ldots, t_m (m\ 0)$，其中 t_i 是一个 S 到 S 的映射函数。一个 $\mathrm{transition}t_i$ 被规格化为 (spre, spost, M, C)。

M&MC 中类的状态图模型定义

- $C = (S, S_0, T)$
- $S = s_1, s_2, \ldots, s_n (n\ 0)$
- $s_i = R_i(v_1), R_i(v_2), \ldots, R_i(v_k)$ 这已经是抽象化的状态了，一个状态可以包含若干种绝对状态，即由状态变量取值决定的状态。
- $T = t_1, t_2, \ldots, t_m (m\ 0)$，其中 t_i 是一个 S 到 S 的映射函数。一个 $\mathrm{transition}t_i$ 被规格化为 (spre, spost, M, C)。

实现者具体定义状态

- 状态的具体定义依赖于实现，由实现者实现一个 purity 方法返回状态。

Write Mode SE	*mode & (1<<FS_WRITE) && maxSize!=0*
Write Mode Normal	*mode & (1<<FS_WRITE) && maxSize==0*
Normal	*curPtr < filePtr*
Overflow1	*curPtr > filePtr + fileSize*
Overflow2	*curPtr>=filePtr && curPtr<=filePtr+fileSize*

状态图模型验证方法

- 范式 1：一个状态是不可达的。
- 范式 2：一个普通状态不能抵达任何其他普通状态。
- 范式 3：存在 null-transition。
- 此外，还存在一些显然的错误，例如允许类从异常中构造，异常可以迁移到普通状态。

状态图模型验证方法

- 范式 1：一个状态是不可达的。
- 范式 2：一个普通状态不能抵达任何其他普通状态。
- 范式 3：存在 null-transition。
- 此外，还存在一些显然的错误，例如允许类从异常中构造，异常可以迁移到普通状态。

状态图模型验证方法

- 范式 1：一个状态是不可达的。
- 范式 2：一个普通状态不能抵达任何其他普通状态。
- 范式 3：存在 null-transition。
- 此外，还存在一些显然的错误，例如允许类从异常中构造，异常可以迁移到普通状态。

状态图模型验证方法

- 范式 1：一个状态是不可达的。
- 范式 2：一个普通状态不能抵达任何其他普通状态。
- 范式 3：存在 null-transition。
- 此外，还存在一些显然的错误，例如允许类从异常中构造，异常可以迁移到普通状态。

状态图模型验证方法

- 范式 1：一个状态是不可达的。
- 范式 2：一个普通状态不能抵达任何其他普通状态。
- 范式 3：存在 null-transition。
- 此外，还存在一些显然的错误，例如允许类从异常中构造，异常可以迁移到普通状态。

状态图可视化问题解决方案

- 根据状态图中状态的数目确定状态的位置分布。
- 根据 (s_{pre}, s_{post}) 将变迁分类。

状态图可视化问题解决方案

- 根据状态图中状态的数目确定状态的位置分布。
- 根据 (s_{pre}, s_{post}) 将变迁分类。

状态图可视化问题解决方案

- 根据状态图中状态的数目确定状态的位置分布。
- 根据 (s_{pre}, s_{post}) 将变迁分类。

Outline

项目背景

- 高层目标：应用于面向对象软件的形式化规格（以及附属的自动化工具、数学推演）。
- 近期目标：可应用于不同层次的模块的形式化规格。
- 此项目研究目标：单个类的规格化、状态图建模和模型验证。

项目背景

- 高层目标：应用于面向对象软件的形式化规格（以及附属的自动化工具、数学推演）。
- 近期目标：可应用于不同层次的模块的形式化规格。
- 此项目研究目标：单个类的规格化、状态图建模和模型验证。

项目背景

- 高层目标：应用于面向对象软件的形式化规格（以及附属的自动化工具、数学推演）。
- 近期目标：可应用于不同层次的模块的形式化规格。
- 此项目研究目标：单个类的规格化、状态图建模和模型验证。

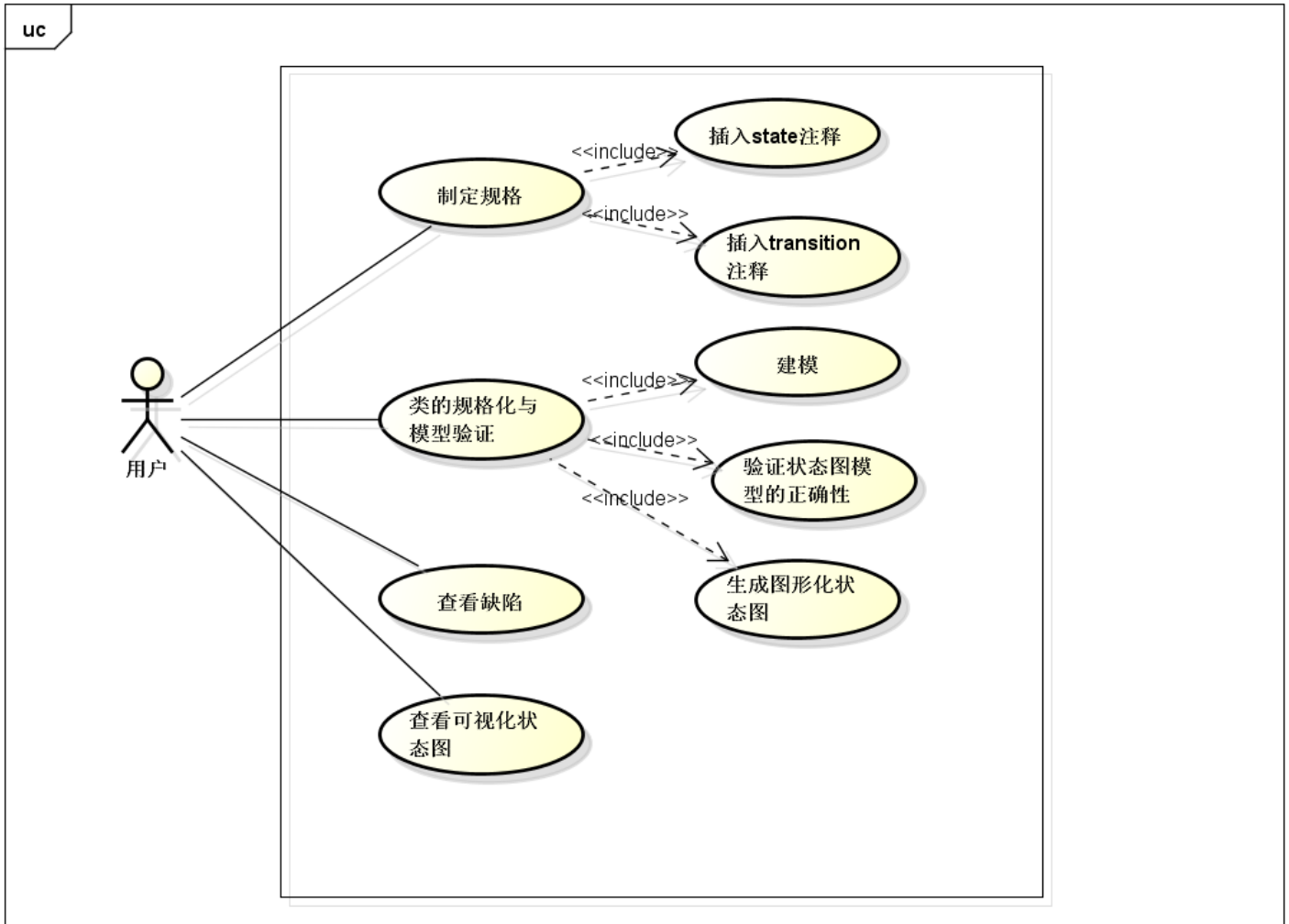
项目背景

- 高层目标：应用于面向对象软件的形式化规格（以及附属的自动化工具、数学推演）。
- 近期目标：可应用于不同层次的模块的形式化规格。
- 此项目研究目标：单个类的规格化、状态图建模和模型验证。

Outline

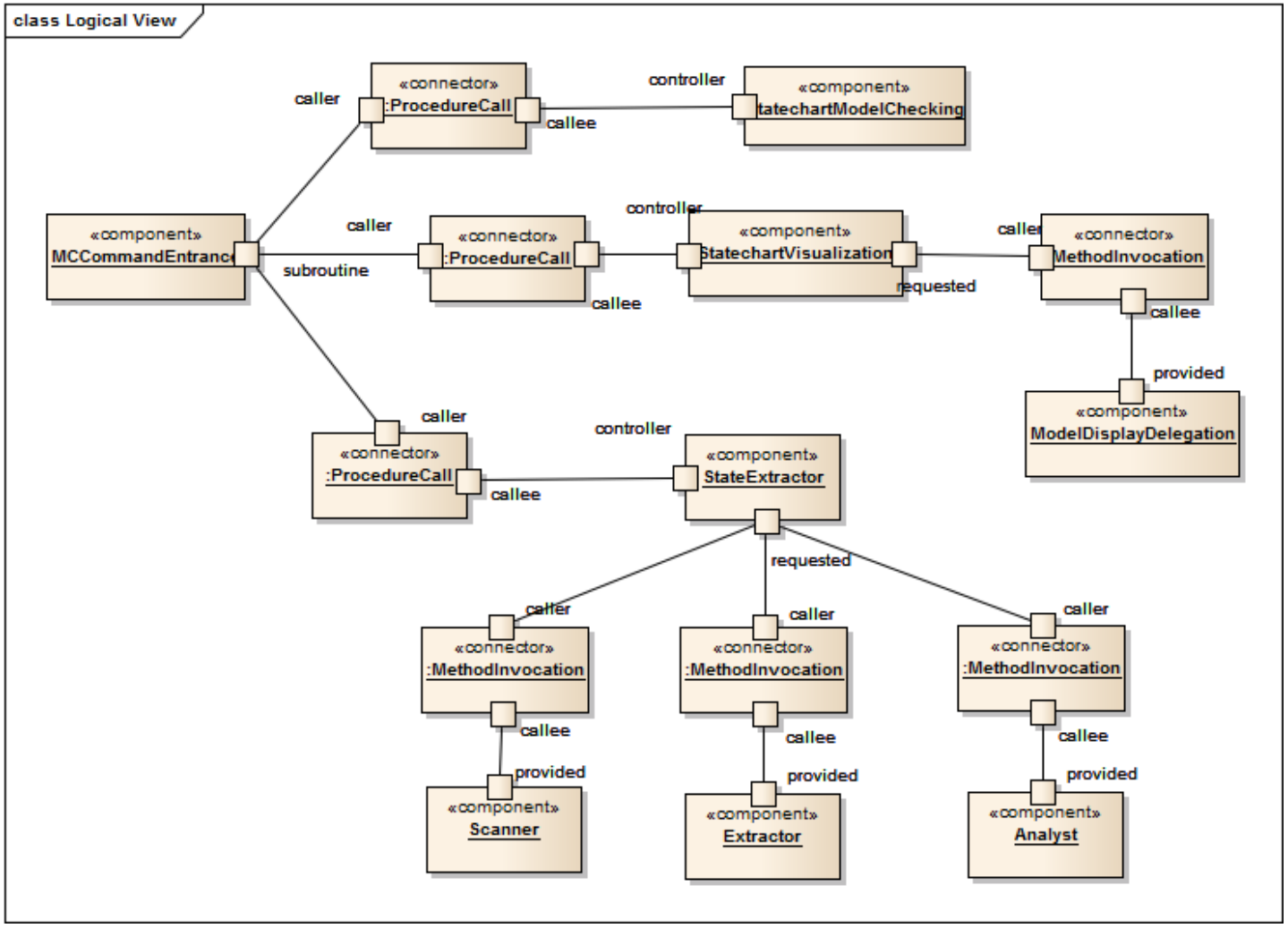
需求分析

- NFR：**可用性与可拓展性**

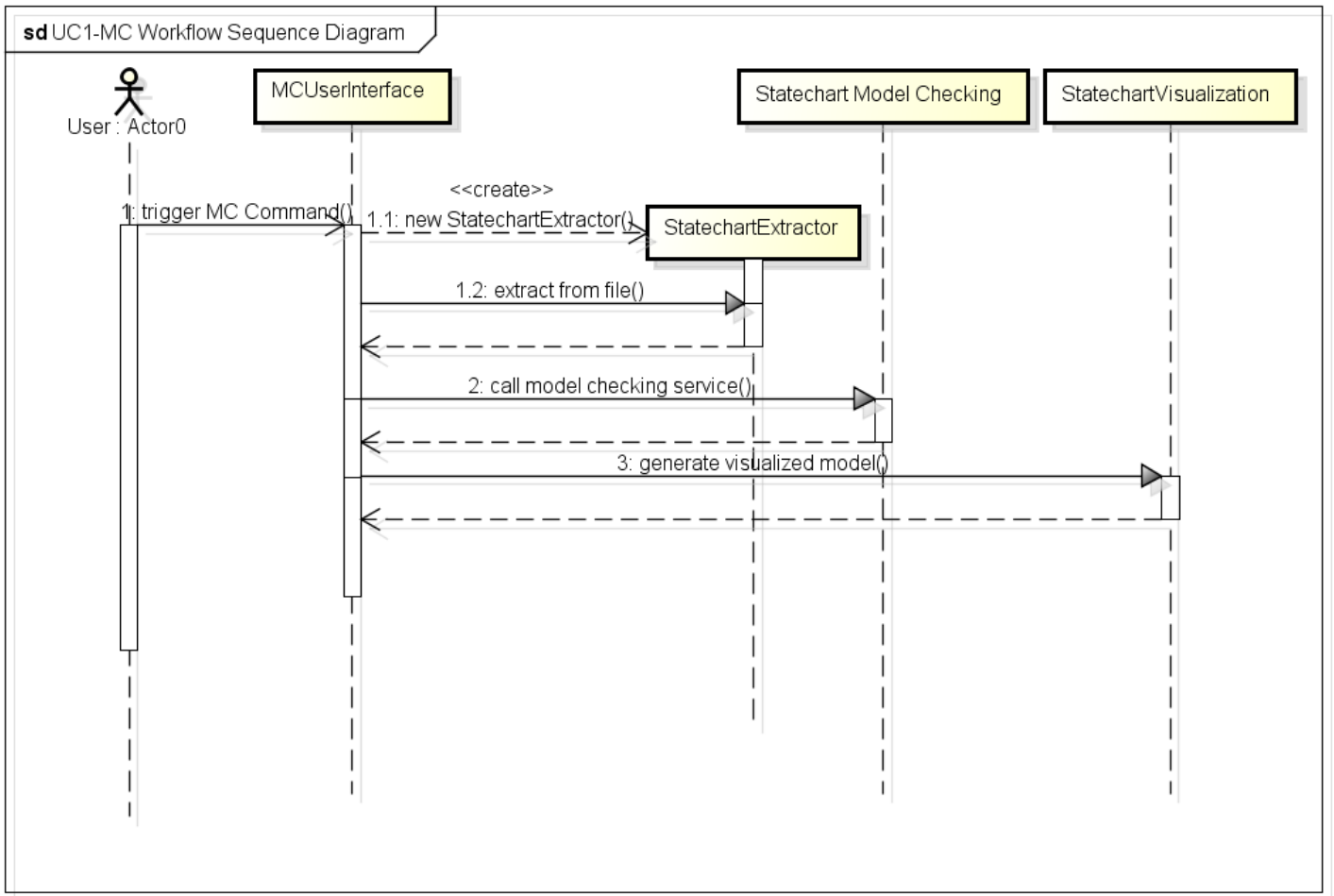


Outline

逻辑视图


class Logical View
«connector»
:ProcedureCall
caller
callee
controller
«component»
tatechartModelChecking
«component»
MCCommandEntranc
caller
subroutine
«connector»
:ProcedureCall
callee
controller
«component»
StatechartVisualization
requested
caller
«connector»
MethodInvocation
callee
provided
«component»
ModelDisplayDelegation
caller
«connector»
:ProcedureCall
callee
controller
«component»
StateExtractor
requested
caller
«connector»
:MethodInvocation
callee
provided
«component»
Scanner
caller
«connector»
:MethodInvocation
callee
provided
«component»
Extractor
caller
«connector»
:MethodInvocation
callee
provided
«component»
Analyst

协作视图


sd UC1-MC Workflow Sequence Diagram
User : Actor0
MCUserInterface
Statechart Model Checking
StatechartVisualization
<<create>>
1: trigger MC Command()
1.1: new StatechartExtractor()
StatechartExtractor
1.2: extract from file()
2: call model checking service()
3: generate visualized model()
powered by astah*

Outline

概述

- StatechartVisualization+MCUserInterface 中的 VisualizedStatechartView 和 MCCommandEntrance。
- 其中 StatechartVisualization 核心部分由 CurveModel 实现，ModelDisplayDelegation 作为中间层将其与 VisualizedStatechartView 与 MCCommandEntrance 这两个模块相连接

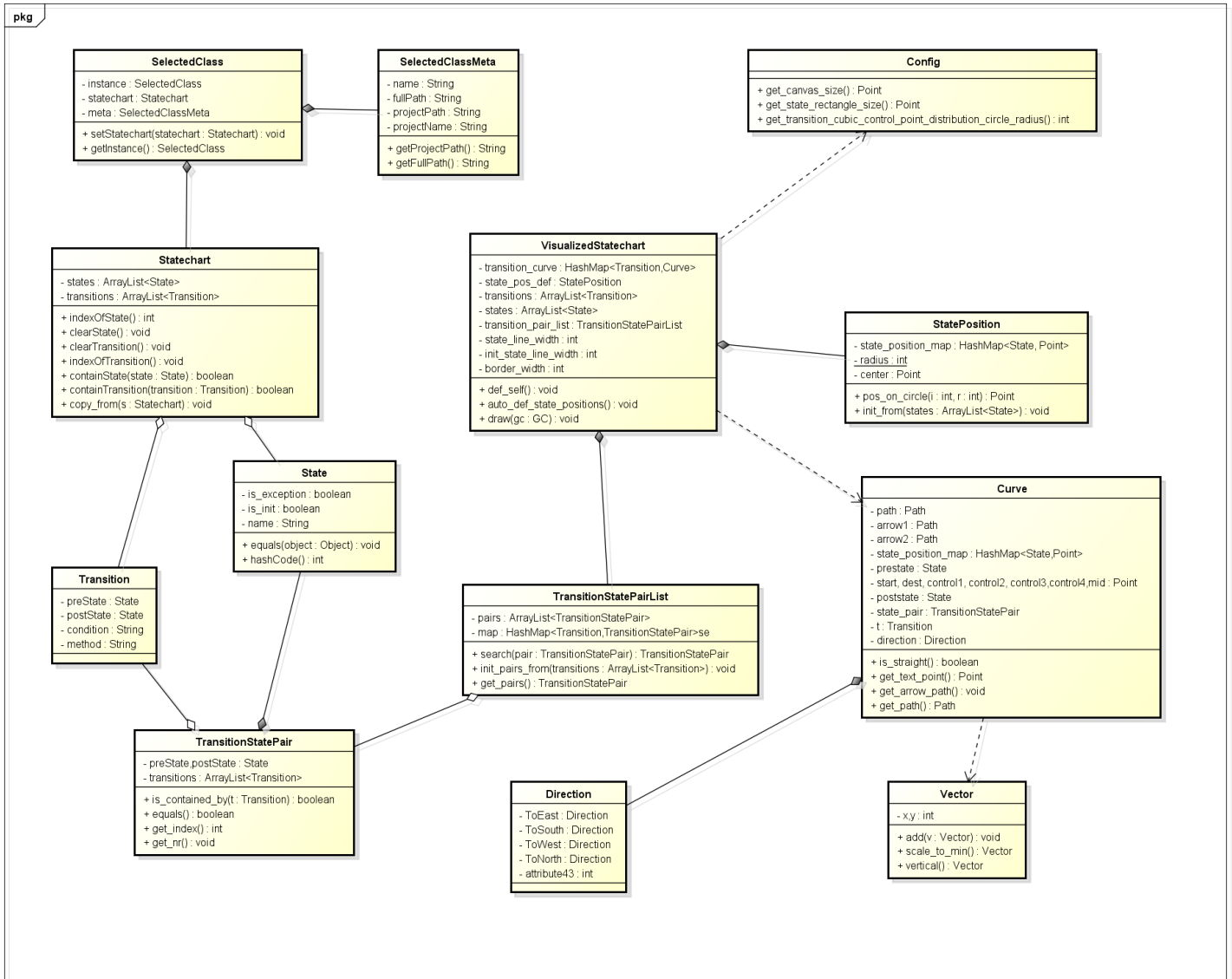
概述

- StatechartVisualization+MCUserInterface 中的 VisualizedStatechartView 和 MCCommandEntrance。
- 其中 StatechartVisualization 核心部分由 CurveModel 实现，ModelDisplayDelegation 作为中间层将其与 VisualizedStatechartView 与 MCCommandEntrance 这两个模块相连接

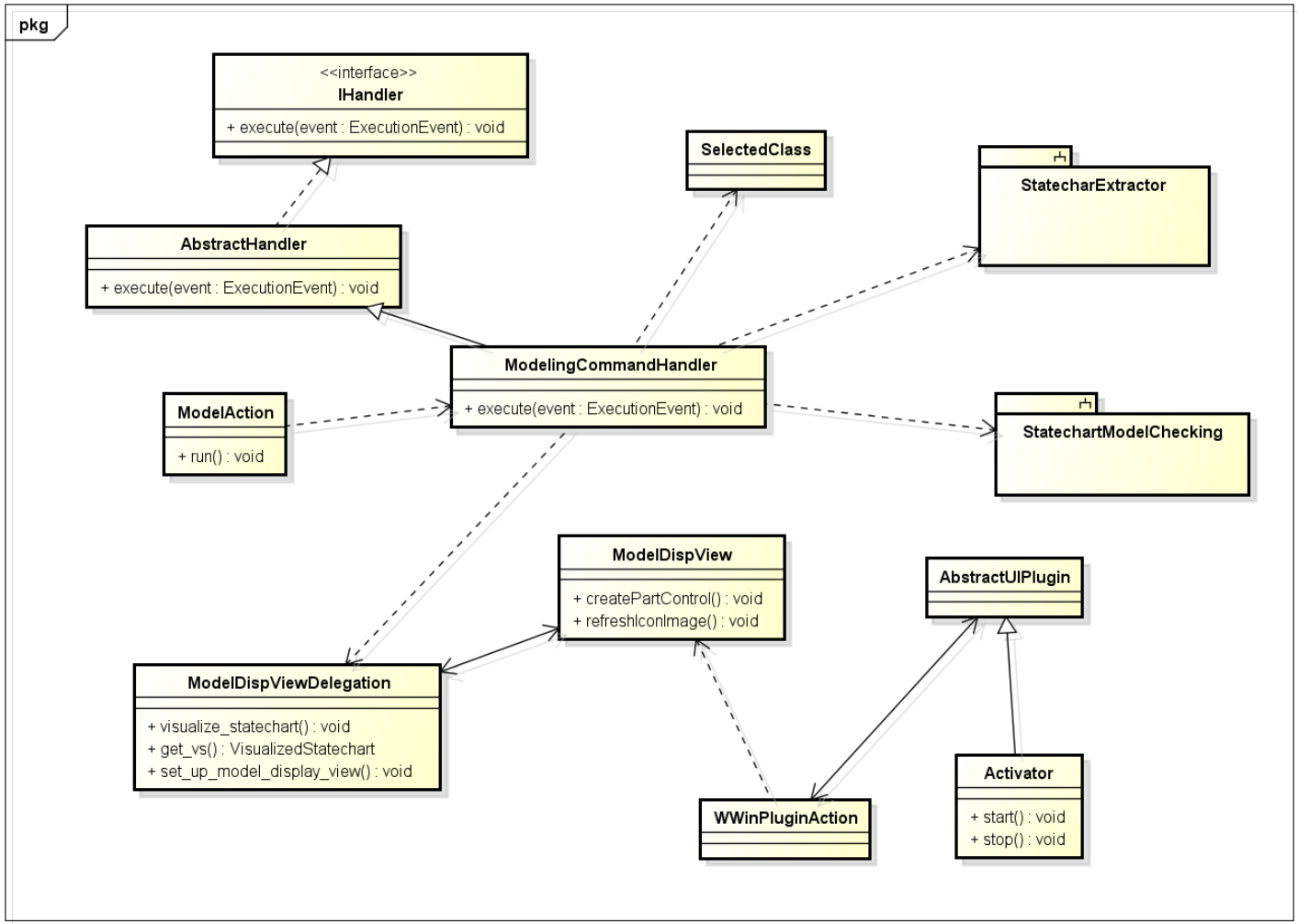
概述

- StatechartVisualization+MCUserInterface 中的 VisualizedStatechartView 和 MCCommandEntrance。
- 其中 StatechartVisualization 核心部分由 CurveModel 实现，ModelDisplayDelegation 作为中间层将其与 VisualizedStatechartView 与 MCCommandEntrance 这两个模块相连接

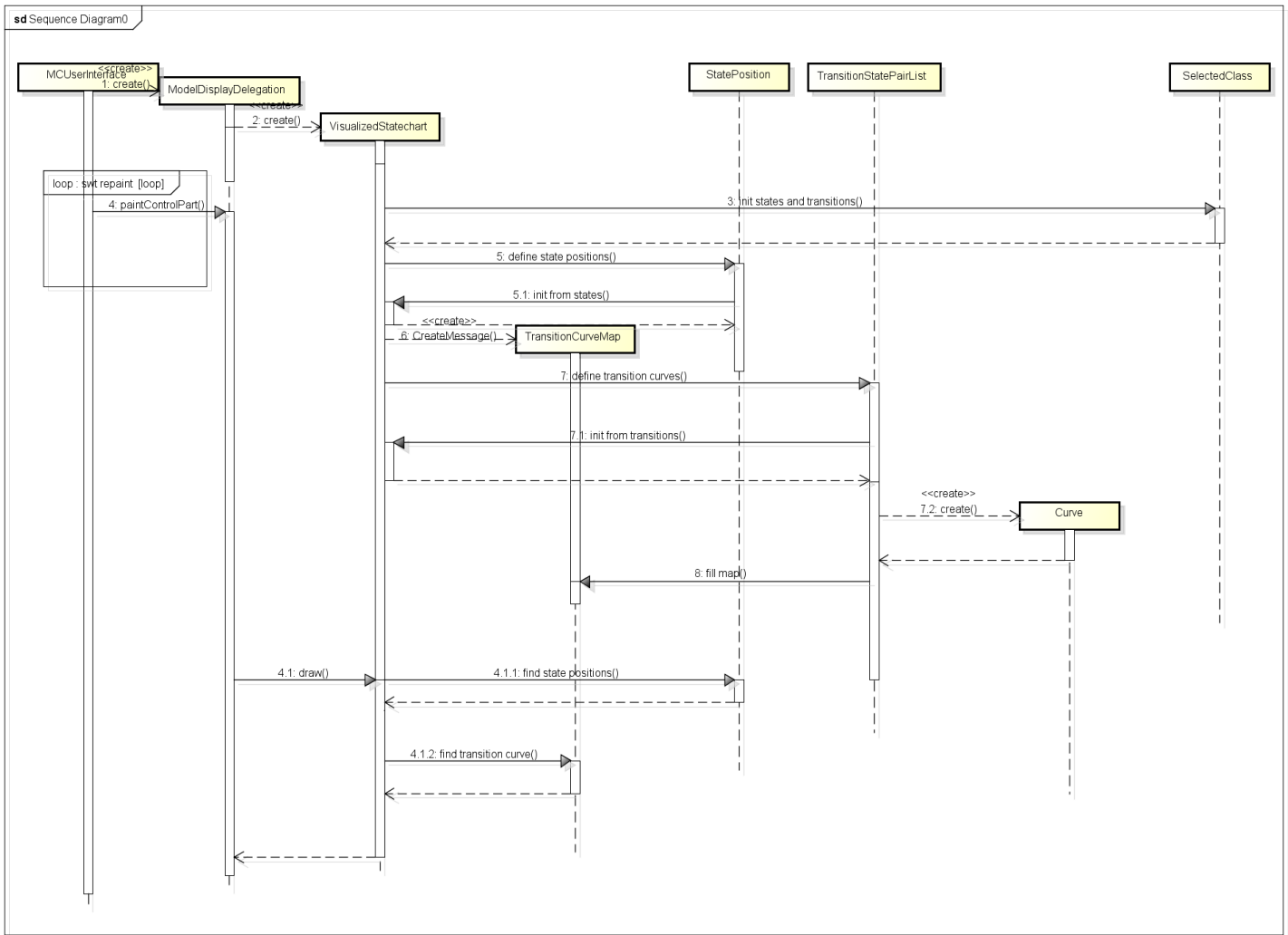
CurveModel 模块



MCCommandEntrance, VisualizedStatechartView 以及中间件 ModelDisplayDelegation



CurveModel、ModelDisplayDelegation 和 MCUserInterface 的入口组件调用顺序



Outline

主要步骤

```
        determine_start_and_dest_points();//确定起始点
        determine_control_points();//确定贝塞尔曲线的控制点
        //首先让曲线经过control1和control2的控制抵达中间点
        path.moveTo(start.x, start.y);
        path.cubicTo(control1.x, control1.y, control2.x, control2.y,
mid.x,mid.y);
        //然后让曲线从中间点通过control3和control4抵达终点。
        path.cubicTo(control3.x, control3.y, control4.x, control4.y, dest.x,
dest.y);
        determine_arrows();//确定箭头路径
```

重要步骤算法

- 根据前置状态到后置状态的连线的方向决定起始点位置
- 确定贝塞尔曲线的控制参数：
  - Curve 模块中使用的变迁路径曲线为 2 条贝塞尔曲线的拼接版本。
  - 控制参数：control1、control2、control3、control4、mid、start、dest。
  - control1 位于以起始点为圆心的一个圆上，所有（前置状态，后置状态）相同的变迁的 control1 都均匀分布在这个圆上，保证变迁曲线在起始点处的切线斜率的分布是均匀的。
  - 同理所有同类 transition 的 control4 也均匀分布在终点为圆心的一个圆上。

重要步骤算法

- 根据前置状态到后置状态的连线的方向决定起始点位置
- 确定贝塞尔曲线的控制参数：
  - Curve 模块中使用的变迁路径曲线为 2 条贝塞尔曲线的拼接版本。
  - 控制参数：control1、control2、control3、control4、mid、start、dest。
  - control1 位于以起始点为圆心的一个圆上，所有（前置状态，后置状态）相同的变迁的 control1 都均匀分布在这个圆上，保证变迁曲线在起始点处的切线斜率的分布是均匀的。
  - 同理所有同类 transition 的 control4 也均匀分布在终点为圆心的一个圆上。

重要步骤算法

- 根据前置状态到后置状态的连线的方向决定起始点位置
- 确定贝塞尔曲线的控制参数：
  - Curve 模块中使用的变迁路径曲线为 2 条贝塞尔曲线的拼接版本。
    - 控制参数：control1、control2、control3、control4、mid、start、dest。
    - control1 位于以起始点为圆心的一个圆上，所有（前置状态，后置状态）相同的变迁的 control1 都均匀分布在这个圆上，保证变迁曲线在起始点处的切线斜率的分布是均匀的。
    - 同理所有同类 transition 的 control4 也均匀分布在终点为圆心的一个圆上。

重要步骤算法

- 根据前置状态到后置状态的连线的方向决定起始点位置
- 确定贝塞尔曲线的控制参数：
  - Curve 模块中使用的变迁路径曲线为 2 条贝塞尔曲线的拼接版本。
  - 控制参数：control1、control2、control3、control4、mid、start、dest。
  - control1 位于以起始点为圆心的一个圆上，所有（前置状态，后置状态）相同的变迁的 control1 都均匀分布在这个圆上，保证变迁曲线在起始点处的切线斜率的分布是均匀的。
  - 同理所有同类 transition 的 control4 也均匀分布在终点为圆心的一个圆上。

重要步骤算法

- 根据前置状态到后置状态的连线的方向决定起始点位置
- 确定贝塞尔曲线的控制参数：
  - Curve 模块中使用的变迁路径曲线为 2 条贝塞尔曲线的拼接版本。
  - 控制参数：control1、control2、control3、control4、mid、start、dest。
  - control1 位于以起始点为圆心的一个圆上，所有（前置状态，后置状态）相同的变迁的 control1 都均匀分布在这个圆上，保证变迁曲线在起始点处的切线斜率的分布是均匀的。
  - 同理所有同类 transition 的 control4 也均匀分布在终点为圆心的一个圆上。

重要步骤算法

- 根据前置状态到后置状态的连线的方向决定起始点位置
- 确定贝塞尔曲线的控制参数：
  - Curve 模块中使用的变迁路径曲线为 2 条贝塞尔曲线的拼接版本。
  - 控制参数：control1、control2、control3、control4、mid、start、dest。
  - control1 位于以起始点为圆心的一个圆上，所有（前置状态，后置状态）相同的变迁的 control1 都均匀分布在这个圆上，保证变迁曲线在起始点处的切线斜率的分布是均匀的。
  - 同理所有同类 transition 的 control4 也均匀分布在终点为圆心的一个圆上。

重要步骤算法

- 根据前置状态到后置状态的连线的方向决定起始点位置
- 确定贝塞尔曲线的控制参数：
  - Curve 模块中使用的变迁路径曲线为 2 条贝塞尔曲线的拼接版本。
  - 控制参数：control1、control2、control3、control4、mid、start、dest。
  - control1 位于以起始点为圆心的一个圆上，所有（前置状态，后置状态）相同的变迁的 control1 都均匀分布在这个圆上，保证变迁曲线在起始点处的切线斜率的分布是均匀的。
  - 同理所有同类 transition 的 control4 也均匀分布在终点为圆心的一个圆上。

重要步骤算法

- 下一步是确定一个曲线必然经过的点，用于添加文字信息标注曲线方法名、守护条件等相关属性，这个点称为曲线中点。曲线中点被定义为 control1 和 control4 的中点，它是第一条贝塞尔曲线的终点和第二条贝塞尔曲线的起点。
  - 最后确定 control2 和 control3，这两个控制参数不应该扭曲曲线的方向，因此分别取 control1 和 mid 的中点，control4 和 mid 的中点。
- 生成曲线路径
- 确定箭头，过终点与 control4 的直线就是位于终点处的切线。

重要步骤算法

- 下一步是确定一个曲线必然经过的点，用于添加文字信息标注曲线方法名、守护条件等相关属性，这个点称为曲线中点。曲线中点被定义为 control1 和 control4 的中点，它是第一条贝塞尔曲线的终点和第二条贝塞尔曲线的起点。
- 最后确定 control2 和 control3，这两个控制参数不应该扭曲曲线的方向，因此分别取 control1 和 mid 的中点，control4 和 mid 的中点。

- 生成曲线路径
- 确定箭头，过终点与 control4 的直线就是位于终点处的切线。

重要步骤算法

- 下一步是确定一个曲线必然经过的点，用于添加文字信息标注曲线方法名、守护条件等相关属性，这个点称为曲线中点。曲线中点被定义为 control1 和 control4 的中点，它是第一条贝塞尔曲线的终点和第二条贝塞尔曲线的起点。
- 最后确定 control2 和 control3，这两个控制参数不应该扭曲曲线的方向，因此分别取 control1 和 mid 的中点，control4 和 mid 的中点。

- 生成曲线路径
- 确定箭头，过终点与 control4 的直线就是位于终点处的切线。

重要步骤算法

-
  - 下一步是确定一个曲线必然经过的点，用于添加文字信息标注曲线方法名、守护条件等相关属性，这个点称为曲线中点。曲线中点被定义为 control1 和 control4 的中点，它是第一条贝塞尔曲线的终点和第二条贝塞尔曲线的起点。
  - 最后确定 control2 和 control3，这两个控制参数不应该扭曲曲线的方向，因此分别取 control1 和 mid 的中点，control4 和 mid 的中点。
- 生成曲线路径
- 确定箭头，过终点与 control4 的直线就是位于终点处的切线。

重要步骤算法

-
  - 下一步是确定一个曲线必然经过的点，用于添加文字信息标注曲线方法名、守护条件等相关属性，这个点称为曲线中点。曲线中点被定义为 control1 和 control4 的中点，它是第一条贝塞尔曲线的终点和第二条贝塞尔曲线的起点。
  - 最后确定 control2 和 control3，这两个控制参数不应该扭曲曲线的方向，因此分别取 control1 和 mid 的中点，control4 和 mid 的中点。
- 生成曲线路径
- 确定箭头，过终点与 control4 的直线就是位于终点处的切线。