# 一、结构设计 （何）

# 二、模块功能设计

## 建模模块

## 验证模块

## 代码生成模块

# 三、信息处理设计（输入、输出）（何）

# 四、关键数据结构设计

## Model类

核心的底层数据结构，用于存储由xml文件生成的模型组件（包括类图、状态机、序列图等）。

### 1.1 属性：

list<Process\*> processes;

list<Property\*> properties;

list<InitialKnowledge\*> initialKnowledges;

SequenceDiagram\* sequenceDiagram;

其中processes属性用于存储Model模型中的所有进程；properties用于存储需要验证的安全属性；initialKnowledges用于存储各个process中的初始知识，sequenceDiagram用于存储序列图。

### 1.2 方法：

## Process类

Model类中核心的数据结构，用于存储协议中进程的相关信息，包括属性、方法、信道、有限状态机等。

### 2.1 属性：

list<Attribute\*> attributes;

list<Method\*> methods;

list<Signal\*> signals;

FiniteStateMachine\* fsm;

Model\* model;

其中attributes是用于存储Process中定义的所有属性，我们支持原生的int、bool数据类型，用户还可以根据需求自己定义数据类型；methods是用于存储Process中定义的所有方法；signals是用于存储Process中定义的所有信道，信道是各进程之间用于通信的通道；fsm是Process对应的有限状态系统，用来定义Process的行为；model是Process所属的模型。

### 2.2 方法：

## FiniteStateMachine类

该数据结构用于存储进程行为的有限状态机，状态机中的每条转移表示从其源状态到目标状态所要进行的条件判断以及行为操作。

### 3.1 属性：

list<Vertex\*> vertices;

list<Edge\*> edges;

Vertex\* startVertex;

Process\* process;

其中vertices是用于存储FiniteStateMachine中的状态集合；edges用于存储FiniteStateMachine中所有的边集合，每条边上我们运行定义一个Guard语句以及若干个Action语句，其中Guard语句用于根据条件判断是否执行后面的语句，Action语句用于对属性的操作；startVertex是用于存储FiniteStateMachine中的初始状态；process是其所属的进程。

### 3.2 方法：

## Xml相关类

Xml类定义XML在内存中的数据结构。Dtd类定义XML的文档头，其中定义了XML文件的元信息。XmlNode类定义了XML树结构的结点。Xml2Model工具类用于实现Xml数据结构和Model数据结构的相互转换。

### 4.1 属性：

XmlNode\* root;

Dtd\* dtd;

这是XML类的属性。其中root记录XML结构树的树根，dtd记录了XML的文档头。

string name;

map<string, string> properties;

list<XmlTree\*> children;

bool isLeaf;

string content;

这是XmlNode类的属性。其中name记录了当前结点的名称，properties记录了当前结点的属性-值的映射，children记录了当前结点的直接孩子列表，isLeaf指示当前结点是否是叶子结点，如果是叶子结点，在content字段中记录结点的字符串信息。

### 4.2 方法：

static Model\* forward(const Xml& xml);

static Xml\* backward(const Model& model);

其中forward用于将Xml数据结构转换为Model数据结构，backward用于将Model转换为Xml数据结构。

## Translator类（王）

## CodeGenerator类（李）

该类是代码生成模块的抽象类。

包含的属性有：Model\* model;

包含的方法有：virtual void generateCode(std::string path)

该虚函数定义了生成代码的方法接口，被SimulationGenerator和ImpleCodeGenerator实现。

**8.1 SimulationGenerator类**

该类用于生成可以模拟系统运行过程的C代码，生成的代码中的方法未被实现。

**8.1.1 重要方法**

std::string generateHeaderFile(const Process\* proc):

用于生成.h文件，文件中包括需要使用的库函数引用，以及需要用到的生成的.h文件和通讯需要用到的部件。

std::string generatePragmas(const FiniteStateMachine\* fsm):

生成Pragma，其中包括对自动机的状态的define。

std::string generateMain(const FiniteStateMachine\* fsm):

生成Main函数的部分，其中重要的是用generateFSMCode方法和Process中的FSM生成自动机的迁移，自动机的运行过程是一个循环，记录当前所在的状态，并根据是否满足迁移的guard进行自动机的迁移，并执行迁移上的动作。

std::string generateProcMethods(const Process\* proc):

生成proc中的方法的代码，此处生成时并不对方法进行实现

std::generateSimCCode(std::string path, std::list<Process\*> procs):

生成所有的代码文件到文件目录path中，该方法对procs中的每一个进程都生成.h和.c文件到目录path。

**8.2 ImpleCodeGenerator类**

该抽象类为有具体实现代码的类的设计提供接口。

**8.2.1 重要方法**

virtual std::string generateDecryptAlgorithm()=0;

生成解密算法的纯虚函数。

virtual std::string generateEncryptAlgorithm()=0;

生成加密算法的纯虚函数。

virtual void generateCode(std::string path, Process\* proc):

对一个Process最终生成其源代码的方法，并将文件输出到目录path。

**8.2.2 说明**

ImpleCodeGenerator还应有对应的子类实现，例如CPPCodeGenerator，在这些类中需要维持一个插件的list，生成加密解密算法的具体部分需要调用相应的插件。

# 五、接口设计

## 内部接口设计

### 1.1 建模模块与模型接口 （刘）

### 1.2 验证模块与模型接口 （王）

### 1.3 代码生成模块与模型接口 （李）

## 外部接口设计 （何）

# 六、执行序列设计 （何）