## 1.3 类

//概述//extends哪些类有，继承的分别是什么

### 连续原子类

1.简介

连续原子类continuous类可对连续模型进行建模。连续原子类本身不可再拆分，在仿真中是最小的结构单元。但这并不意味着连续原子类仅仅能建立“小”模型，根据建模抽象层次不同，连续原子类可对不同“大小”的模型进行建模。例如,对于某一电路模型的描述，既可以用continuous类去描述各个元器件，再将它们来连接起来实现电路建模，也可以直接用一个连续原子类对整个电路进行建模，只要满足整个电路的等式约束关系即可。

2.基本结构

continuous类包含头部份、定义部分、等式部分。以如下例子说明

头部份包括导入外部模型（结构关键字import）以及继承外部模型（结构关键字extends）两部分内容。Import指的是下文需要对外部类进行实例化，在continuous类中，import后的模型类常常是connector类，function类，record类以及continuous类，其中导入的continuous类只能用于继承，connector类用于后面port部分的实例化，function类用于，record类用于。Extends可用于对于模型的继承，在continuous类中，继承的对象只能是continuous类。比如下例中，Model\_B必须为continuous类。

定义部分描述了continuous类的属性，包括parameter、value、port三个结构关键字引领的部分。其中，parameter部分定义了continuous模型的恒定属性，等号后为属性的值，属性在仿真中不会更改，是个常数，比如恒温电阻的阻值；value部分定义了模型在仿真过程中的变量，如果变量后有等号，则等号后为变量的初始值，如果变量恒定，则可在其前加上限定词constant；port部分定义了模型的端口部分即输入输出部分，这部分主要有两种表达形式，一种是用input/output作为限定词修饰的量，另一种是对connector类的实例化（connector类将在后文详细介绍）例如下例中就定义了connector类Model\_A的实例model2。

等式部分描述了continuous类的行为。该部分以equation作为关键字，所使用的变量以及常量都是在定义部分定义过的量。der()作为内置微分函数表示对变量的微分运算，此外，此部分还可以应用if，for等结构体来描述模型的行为。注意，等式部分的等于号表示的是方程中的一种相等的约束关系，并不表赋值。

各个部分的缺省情况介绍。下例中给出的是continuous全部的结构。当然在实际建模中，一个模型可能不会包括所有部分。各个结构关键字引领的部分根据模型需要去编写。如没有该部分，则连同结构关键字一同省略。最常见的用法比如：某continuous部分仅仅具有头部份和定义部分，而缺省等式部分，这样的continuous常常作为继承的对象，表示一些模型的公共定义部分，比如电路模型中的一端口、二端口等。

#示例代码

continuous Model\_name

import Model\_A;

import Model\_B;

extends Model\_B;

parameter:

real a=1;

bool b=false;

value:

real c=a;

real d=0;

real x;

constant real g=9.0;

port:

input real in1;

output bool out1;

Model\_A model2;

equation:

der(x)=a;

if…

for…

end;

3.建模例子：#inital equation的作用，与value

一阶系统模型：

continuous FirstOrder

parameter:

real k=1;

port:

input real x=0;

output real y;

equation:

der(y)+k\*x=0;

end;

一端口模型：

continuous OnePort

import X.electric.Interfaces.PositivePin;//PositivePin是connector类,详见1.3.6

import X.electric.Interfaces.NegativePin;//negativePin是connector类

value:

real v;

real i;

port:

PositivePin p;

NegativePin n;

equation:

v=p.v-n.v;

p.i+n.i=0;

i=p.i;

end;

电阻模型：

continuous Resistor

import X.electric.Interfaces.Oneport;

extends Oneport;

parameter:

real R=1;

equation:

v=R\*i;

end;

水箱模型：

### 1.3.2离散原子类#对数字电路的支持

1.简介

离散原子类discrete类可对离散模型进行建模。离散原子类本身不可再分，是离散模型最小的仿真单元。与连续原子类类似，最小的仿真单元并不意味着描述能力上离散原子类仅仅能建立“小”模型。例如，对于某复杂事件的建模，既可以将复杂事件拆解成简单事件再加以连接，也可以直接对复杂事件整体进行建模，只要逻辑关系满足即可。

2.基本结构（header等形式规范，按section展开，必须与非必须）

以下面的示例代码来说明discrete类的基本结构及组成。

discrete类包含头部，定义部分与状态部分。

头部份可以对外部类进行导入或者是进行继承操作。

定义部分描述了discrete类的属性，包含parameter、value、port等，其中parameter与value与continuous类中的功能相似，port中不再含有connector连接类的实例化，并且需要在input/output前加上限定词event来表示事件的输入输出。

状态部分是离散原子类独有的部分，以关键词state引领该部分内容。state部分由多个状态机组成。如上例所示，包含三个状态机，初始状态机name\_0，name\_1,name\_2。初始状态机是每个state部分必须含有的状态机，表示离散原子类的初始进入状态，由关键字inital state加初始状态机名字组成，上例中，初始状态机名为name\_0。其余状态机由state加状态机名称组成，例如状态机name\_1,name\_2。

下面介绍状态机内部的结构。

状态机内部以when引领，表示在条件的触发下，状态机的行为，例如向外输出，状态切换等，给变量赋值，也可以包含if条件语句，循环语句等等。其中

When entry() then

…

end;

上述以entry()函数为条件的when结构是每一个状态空间必须要有的，表示状态机进入该状态后的初始行为。

state部分用到的函数或者结构有：

结构：

when结构

When condition then

…

End；

输出结构

Out：

  Send()；

函数：

Receive()

Receive函数定义了哪些端口接收到消息时该事件行为会被触发。Receive语句中可以包括一个或多个参数，每一个参数都必须对应到Atomic类中声明的输入接口。

Entry()

Entry函数用于表示进入一个状态，无参数，Entry函数在DEVS中并未进行相关的定义，是X语言中为了表述上的便利所做出的改变。

Send()

Send函数表示输出到对应端口，send函数包括两个参数，第一个参数为模型的输出端口，第二个参数为输出到端口的数值。

Trasition()

在X语言中，使用transition函数表示状态的转移，transition函数的参数必须为State模块中定义的多个状态中的一个，且状态可以实现自循环，既可以实现从一个状态转移到他自己。

#示例代码

discrete And

import Model\_A;

extends Model\_A;

parameter:

real a=0;

value:

bool h=Ture;

port:

event output bool out0;

event input bool in0;

event input bool in1;

state：

initial state name\_0

When entry() then

end;

when receive(in0,in1) then

trasition(outer);

end;

end;

state name\_1

When entry() then

end;

 When entry() then

if in0=true and in1=ture then

out:

send(out0,ture);

else:

 out:

    send(out0,false);

end；

 end；

end;

state name\_2

...

end;

...

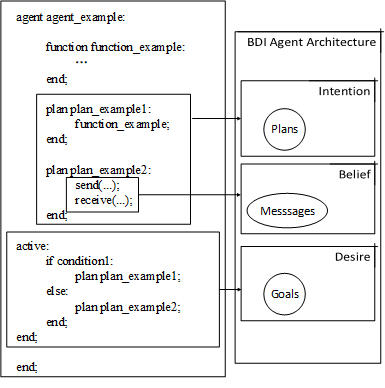
end；

3.建模例子：看门狗、 （对标上面）

### 1.3.3智能体类

1.简介

X语言中智能体类的设计遵循BDI(Belief-Desire-Intention)架构，保留了goal指导plan的执行的过程，而将环境感知等其他内容留给了用户自己去定义，以提升语言的扩展性。在利用Agent类进行建模时，整个结构可以分为两部分，分别是definition部分和action部分。definition部分用来初始化参数和变量的值，以及函数和计划的声明也在该部分完成；action部分用来控制计划的执行，以及设置智能体仿真的开始和终止条件。

agent类文本的结构与BDI的对应关系如下图所示，左侧是agent类文本的结构，右侧BDI架构图。计划对应于BDI架构中的Intention部分，执行部分定义的内容对应于goal部分，而Belief部分则被融入到了整个智能体与环境交互感知的过程中。在保留了原始BDI架构特点的基础上，与X语言的语法语义进行了融合，赋予了X语言智能化特性。

2.基本结构（header等形式规范，按section展开，必须与非必须）

2.1 header

X语言中的智能体类需要用agent关键字声明：

agent counter

2.2 definition

Definition部分包含了整个agent类中涉及的所有变量和方法，主要可能包括以下几种：

1）parameter

用于定义这个agent类传入的参数，以实例化该agent类：

parameter String agentname

2）plan

用于定义agent类中的核心动作集合，每个plan又包含了两个部分，分别是value和action部分：

Value用于定义用于plan内部的局部参数：

 plan plan1

    value:

Integer perception = 0;

String temp[1] = ["atomictest"];

Action用于定义agent的行为：

plan plan1

    value:

Integer perception = 0;

String temp[1] = ["atomictest"];

    action:

        msg = message{temp, perception + 1};

        send(msg);

        temp[0] = "default";

        perception = receive(false,)[0];

        count = count + 1;

其中，action中包含两个内置函数，专门用于传递agent之间的消息，分别是send(message)和receive(is\_wait):

send(msg);

perception = receive(false,)[0];

3）value

用于定义作用域范围在整个agent类的全局变量，使得agent类内的任何方法都可以对其进行修改

value:

Real count = 5;

Real a = 1;

2.3 action

整个agent类下的action其实就是原始BDI中的Goal，用于引导plan的执行，通过run函数：

 if count == 100 then

        agentover;

end;

    run (plan1);

其中，关键词agentover用于表示该agent仿真的结束

2.4 couple

在定义好了agent的类之后，还需要利用couple将其与其他模型进行连接，相当于实例化的过程。整个couple过程与其他模型的couple过程一致，都需要经过import，实例化，以及connect三个过程。

3.建模例子：（对标上面，各种不同的应用情况）

Counter.xl：

agent counter

parameter String agentname;

    plan plan1

    value:

Integer perception = 0;

String temp[1] = ["atomictest"];

    action:

        msg = message{temp, perception + 1};

        send(msg);

        temp[0] = "default";

        perception = receive(false,)[0];

        count = count + 1;

end;

value:

Real count = 5;

Real a = 1;

action:

if count == 100 then

        agentover;

end;

    run (plan1);

end;

Agentcouple.xl

couple simulation

record Message

    value:

String from;

        list<String> to;

Integer content;

String protocol;

end;

import counter;

import AtomicTest;

part:

    AtomicTest atomic\_test;

    counter counter\_1(agentname = "agent\_test");

connection:

    connect(atomic\_test.inport,counter\_1.outort);

    connect(counter\_1.inport,atomic\_test.outort);

end;

### 1.3.4耦合类

1.简介

耦合类是X语言的一个重要的类，也是X语言实现仿真级建模与系统建模联通的关键。耦合模型在X语言中承担了连接多领域模型仿真的任务，在耦合模型中，不仅能够连接相同领域（如离散模型），也能够连接不同领域的模型，只需要二者之间的端口能够相互匹配。是X语言中实现复杂系统建模与仿真的主要类工具。

2.基本结构（header等形式规范，按section展开，必须与非必须）

耦合类以关键字couple统领。耦合类包括头部份、属性部分、连接部分等三个部分。

头部份包括导入外部模型（结构关键字import）以及继承外部模型（结构关键字extends）两部分内容。Extends部分的类只能是couple类，import部分根据需要导入外部类，以便继承或者实例化。

属性部分包括parameter、port、part三个部分。Parameter表示couple类的固有属性，为常数。Port表示couple类的接口。Part部分表示couple包含的子模块，内容包含continuous、discrete、couple类的实例化。

连接部分以connection关键字统领，内部以connect的形式连接各个端口。connect连接的两端连接的量的属性必须一致，可以同为单变量，但是需要值的数值类型一致，也可以同为connector类。

例如下例中，导入了connector、continuous、discrete、couple四个模型，并继承了couple\_1。part部分，对于continuous\_1与discrete类进行实例化。port部分，利用导入的connector\_1定义自己的端口。connection部分，将name\_2的p端口与name\_3的n端口相连，将name\_2的n端口与本couple模型的name\_1端口相连，并且，由于name\_1为connector类，可以推测name\_2的p端口p端口也是connector类。

#示例代码

couple Model\_1

import connector\_1;

import continuous\_1;

import discrete\_1;

import couple\_1;

extends couple\_1;

parameter:

real a=1;

port:

connector\_1 name\_1;

part:

continuous\_1 name\_2(am=a);

discrete\_1 name\_3;

connection:

connect(name\_2.p,name\_3.n);

connect(name\_2.n,name\_1);

end;

3.建模例子

### 函数类#函数中input可初始化么

1.简介

2.基本结构（header等形式规范，按section展开，必须与非必须）

函数类以关键字function统领。包含头部份，定义部分以及行为部分。头部份可以导入外部函数类。定义部分包括port与value两个部分，port中定义了函数的输入输出。行为部分以关键字action引领，action部分可以用于描述输入输出之间的关系。

#示例代码

function Model\_1

import fun\_0;

port:

input real x;

output real y;

value:

real z=0;

action:

z=der(x)

  if z>=0 then

y=x+1;

 else

y=x-1;

end;

end;

3.建模例子：（header,）

### 连接器类(连续与离散相连)

1.简介

连接器类是实现组件化建模的关键类，connector是一种让模型与模型交换信息的方法。Connector主要针对物理系统的非因果建模。非因果物理建模方法区分了两类不同的变量-流变量与势变量。一般用connector来表示两个组件端口之间具有的绑定语义或者满足基尔霍夫定理，即，势变量相等，流变量之和等于0。

2.基本结构（flow real ,real）(real)

连接器类以关键字connector统领。只包含value一个部分。value部分可以包含多个不同数据类型的量。一般在建模过程中，连续类会调用连接器类实现非因果建模。

#示例代码

connector PositivePin

value:

real v;

real i;

end;

3.建模例子：（header,）

### 纪录类

1.简介

record类主要用于实现模型数据的分组，适用于模型的参数由多组不同类型的量组成时建立。

2.基本结构

纪录类以关键字record统领。只包含value一个部分。value部分可以包含多个不同数据类型的量。一般在建模过程中，连续类、离散类、函数类都可能调用纪录类来便于描述多组不同类型的量。

#示例代码

record perception

int reward;

list<int> autocar\_position;

list<int>current\_postion;

end;

3.建模例子