UNIVERSIDADE FEDERAL DE OURO PRETO
***CAMPUS* MORRO DO CRUZEIRO**

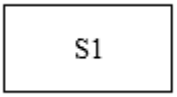
MATHEUS PEIXOTO RIBEIRO VIEIRA - 22.1.4104

SPRINT 01 DE ENGENHARIA DE SOFTWARE 1: PROJETO DA API DO SIMULADOR DE SISTEMAS DINÂMICOS

OURO PRETO
OUTUBRO DE 2023

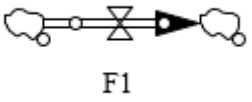
Casos de Uso

1 - Sistema sozinho



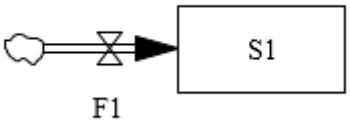

```C/C++
Model m;
System S1;
m.add(&s1);
m.run(100);
```

2 - Fluxo sozinho



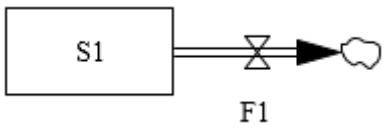

```C/C++
Model m;
Flow F1;
m.add(&F1);
m.run(100);
```

3 - Fluxo sem origem conectado a um sistema



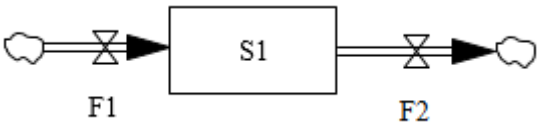

```C/C++
Model m;
Flow f1;
System s1;
m.add(&f1);
m.add(&s1);
f1.setTarget(&S1)
m.run(100);
```

4 - Sistema conectado a um fluxo



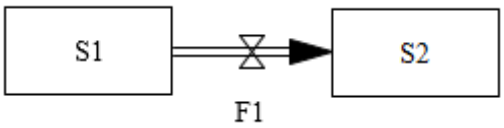

```
C/C++
Model m;
System S1;
Flow F1;
F1.setSource(&S1);
m.add(&S1);
m.add(&F1);
m.run(100);
```

5 - Sistema com um fluxo de entrada e outro de saída




```
C/C++
Model m;
System S1;
Flow F1, F2;
F1.setTarget(&S1);
F2.setSource(&S1);
m.add(&S1);
m.add(&F1);
m.add(&F2);
m.run(100);
```

6 - Dois sistemas conectados por um fluxo

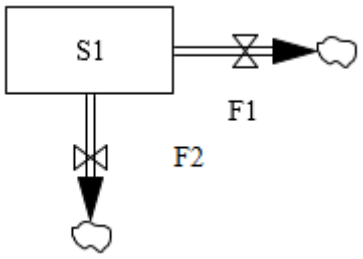


```C/C++
Model m;
System S1, S2;
Flow F1;
F1.setSource(&S1);
F1.setTarget(&S2);
m.add(&S1);
m.add(&S2);
m.add(&F1);
m.run(100);
```

Outra maneira que também seria possível, seria realizar uma sobrecarga no construtor de Flow para colocar o nome do flow, origem e destino, evitando criação de mais linhas, evitando uma poluição do código.

```C/C++
Model m;
System S1, S2;
Flow F1 ("F1", &S1, &S2);
m.add(&S1);
m.add(&S2);
m.add(&F1);
m.run(100);
```

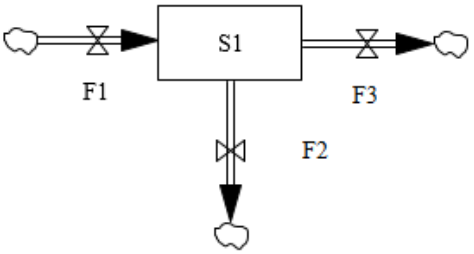
7 - Um sistema com dois fluxos de saída




```C/C++
Model m;
System S1;
Flow F1, F2;
F1.setSource(&S1);
F2.setSource(&S1);
m.add(&S1);
m.add(&F1);
```

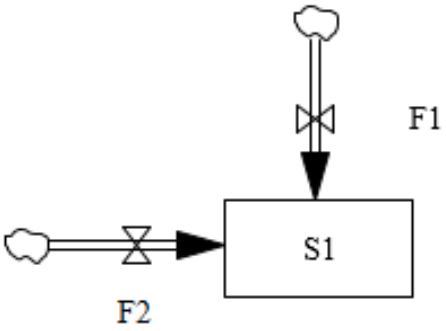
```
m.add(&F2);
m.run(100);
```

8 - Sistema com um fluxo de entrada e múltiplos fluxos de saída



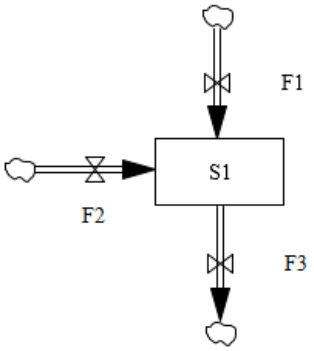

```
C/C++
Model m;
System S1;
Flow F1, F2, F3;
F1.setTarget(&S1);
F2.setSource(&S1);
F3.setSource(&S1);
m.add(&S1);
m.add(&F1);
m.add(&F2);
m.add(&F3);
m.run(100);
```

9 - Sistema recebendo múltiplos fluxos



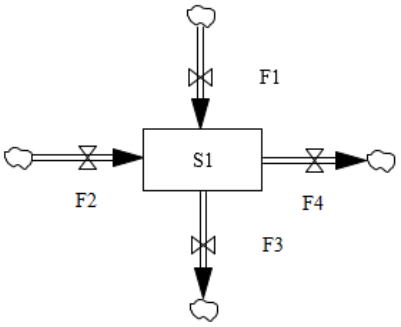
```C/C++
Model m;
System S1;
Flow F1, F2;
F1.setTarget(&S1);
F2.setTarget(&S1);
m.add(&S1);
m.add(&F1);
m.add(&F2);
m.run(100);
```

10 - Sistema com múltiplos fluxos de entrada e um fluxo de saída



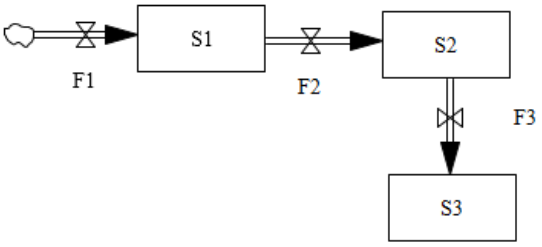

```C/C++
Model m;
System S1;
Flow F1, F2, F3;
F1.setTarget(&S1);
F2.setTarget(&S1);
F3.setSource(&S1);
m.add(&S1);
m.add(&F1);
m.add(&F2);
m.add(&F3);
m.run(100);
```

11 - Sistema com múltiplos fluxos de entrada e múltiplos fluxos de saída




```
C/C++
Model m;
System S1;
Flow F1, F2, F3, F4;
F1.setTarget(&S1);
F2.setTarget(&S1);
F3.setSource(&S1);
F4.setSource(&S1);
m.add(&S1);
m.add(&F1);
m.add(&F2);
m.add(&F3);
m.run(100);
```

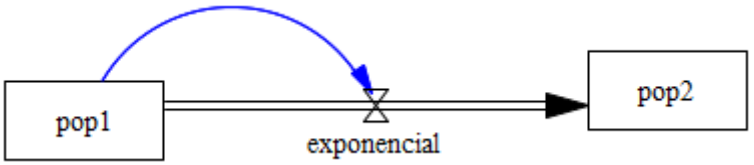
12 - Múltiplos sistemas e fluxos




```
C/C++
Model m;
System S1, S2, S3;
Flow F1, F2, F3;
F1.setTarget(&S1);
F2.setOrigin(&S1);
F2.setTarget(&S2);
```

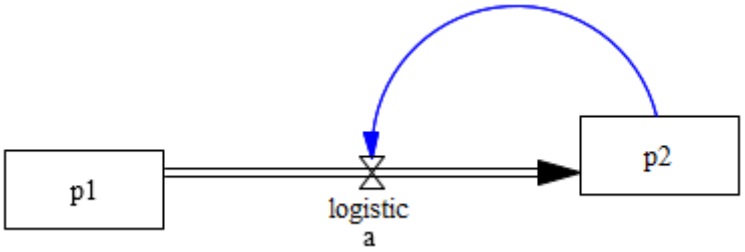
```
F3.setOrigin(&S2)
F3.setTarget(&S3);
m.run(100);
```

Critério de aceitação



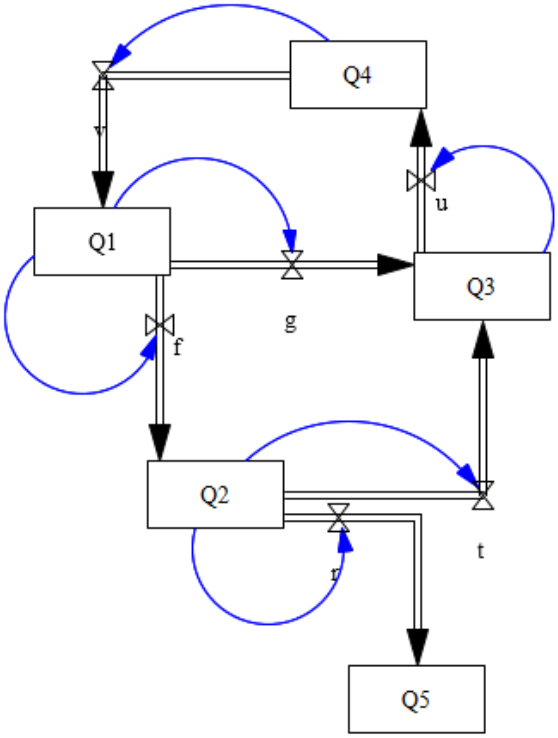
Um grande problema enfrentado durante a formulação dos modelos seria a de como atribuir uma função para o Flow, no exemplo seria a função exponencial que está conectando os sistemas pop1 e pop2. Desta forma, pode-se utilizar uma função anônima que recebe um valor double de entrada e retorna, também, um valor double.

```C/C++
int main(){
    System * pop1 = new System("pop1", 100);
    System * pop2 = new System("pop2", 200);

    Flow * exponencial = new Flow("exponencial")
    exponencial.setOrigin(pop1);
    exponencial.setTarget(pop2);
    exponencial.setEquation([](double input){
        return input * 0.01;
    });

    Model m;
    m.add(pop1);
    m.add(pop2);
    m.add(exponencial);
    m.run(100);

    return 0;
}
```



A função de logística seguirá o mesmo conceito da exponencial.

```C/C++
int main(){
    System * p1 = new System("p1", 100);
    System * p2 = new System("p2", 200);

    Flow logistica;
    logistica.setOrigin(p1);
    logistica.setTarget(p2);
    logistica.setEquation([](double input){
        return 0.01*input*(1 - input / 70)
    });

    Model m;
    m.add(p1);
    m.add(p2);
    m.add(logistica);
    m.run(100);

    return 0;
}
```

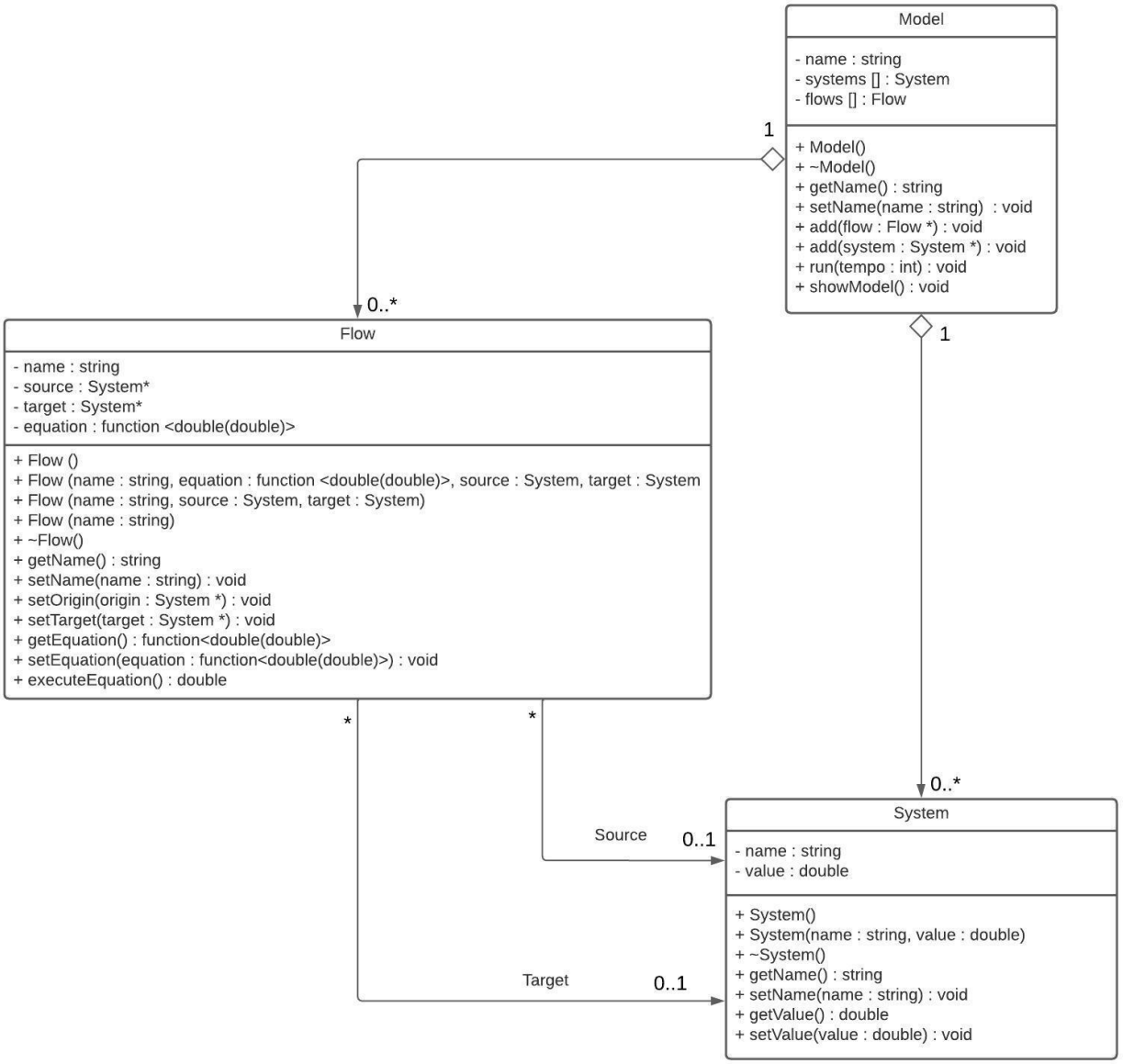


Uma sobrecarga no construtor do Flow pode ser realizada a fim de atribuir, diretamente na instanciação da classe, o nome, equação, origem e destino. Dessa forma, o código da API fica mais legível e enxuto.

```C/C++
int main(){
  System * Q1, * Q2, * Q3, * Q4, * Q5;

  auto equation = [](double input){
    return input * 0.01;
  };

  Flow * f = new Flow("f", equation, Q1, Q2);
  Flow * g = new Flow("g", equation, Q1, Q3);
  Flow * r = new Flow("r", equation, Q2, Q5);
  Flow * t = new Flow("t", equation, Q2, Q3);
  Flow * u = new Flow("u", equation, Q3, Q4);
  Flow * v = new Flow("v", equation, Q4, Q1);

  Model m;
  m.add(Q1);
  m.add(Q2);
  m.add(Q3);
  m.add(Q4);
  m.add(Q5);
  m.add(f);
  m.add(g);
  m.add(r);
```

```
  m.add(t);
  m.add(u);
  m.add(v);

  model.run(100);

  return 0;
}
```

UML


Model
- name : string
- systems [] : System
- flows [] : Flow
+ Model()
+ ~Model()
+ getName() : string
+ setName(name : string) : void
+ add(flow : Flow *) : void
+ add(system : System *) : void
+ run(tempo : int) : void
+ showModel() : void
1
0..*
1
0..*
Flow
- name : string
- source : System*
- target : System*
- equation : function <double(double)>
+ Flow ()
+ Flow (name : string, equation : function <double(double)>, source : System, target : System
+ Flow (name : string, source : System, target : System)
+ Flow (name : string)
+ ~Flow()
+ getName() : string
+ setName(name : string) : void
+ setOrigin(origin : System *) : void
+ setTarget(target : System *) : void
+ getEquation() : function<double(double)>
+ setEquation(equation : function<double(double)>) : void
+ executeEquation() : double
*
*
Source
0..1
Target
0..1
System
- name : string
- value : double
+ System()
+ System(name : string, value : double)
+ ~System()
+ getName() : string
+ setName(name : string) : void
+ getValue() : double
+ setValue(value : double) : void