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

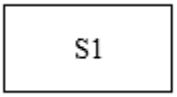
MATHEUS PEIXOTO RIBEIRO VIEIRA - 22.1.4104

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

OURO PRETO
OUTUBRO DE 2023

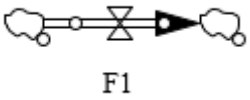
Casos de Uso

1 - Sistema sozinho



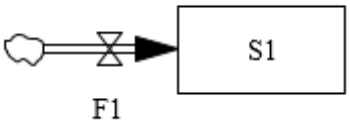

```C/C++
int main(){
    Model m;
    System S1;
    m.add(&S1);
    m.run(1, 100);
    return 0;
}
```

2 - Fluxo sozinho




```C/C++
int main(){
    Model m;
    MyFlow F1;
    m.add(&F1);
    m.run(1, 100);
    return 0;
}
```

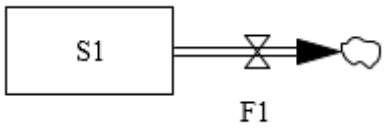
3 - Fluxo sem origem conectado a um sistema




```C/C++
int main(){
    Model m;
    MyFlow f1;
    System s1;
    m.add(&f1);
    m.add(&s1);
```

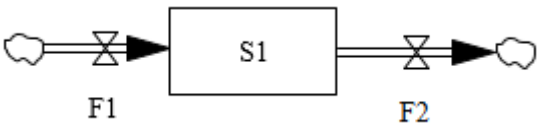
```
    f1.setTarget(&s1);
    m.run(1, 100);
    return 0;
}
```

4 - Sistema conectado a um fluxo




```C/C++
int main(){
    Model m;
    System S1;
    MyFlow F1;
    F1.setSource(&S1);
    m.add(&S1);
    m.add(&F1);
    m.run(1, 100);
    return 0;
}
```

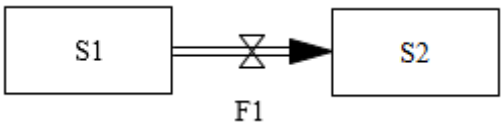
5 - Sistema com um fluxo de entrada e outro de saída




```C/C++
int main(){
    Model m;
    System S1;
    MyFlow F1, F2;
    F1.setTarget(&S1);
    F2.setSource(&S1);
```

```
    m.add(&S1);
    m.add(&F1);
    m.add(&F2);
    m.run(1, 100);
    return 0;
}
```

6 - Dois sistemas conectados por um fluxo



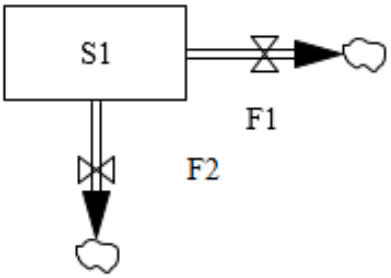

```C/C++
int main(){
    Model m;
    System S1, S2;
    MyFlow F1;
    F1.setSource(&S1);
    F1.setTarget(&S2);
    m.add(&S1);
    m.add(&S2);
    m.add(&F1);
    m.run(1, 100);
    return 0;
}
```

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++
int main(){
    Model m;
    System S1, S2;
    MyFlow F1 ("F1", &S1, &S2);
    m.add(&S1);
    m.add(&S2);
    m.add(&F1);
    m.run(1, 100);
```

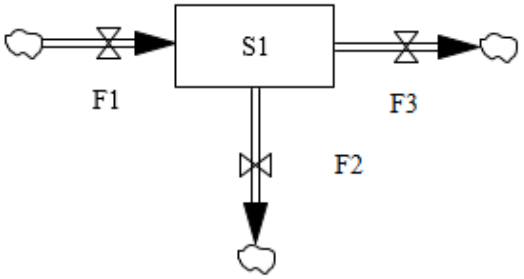
```
    return 0;
}
```

7 - Um sistema com dois fluxos de saída



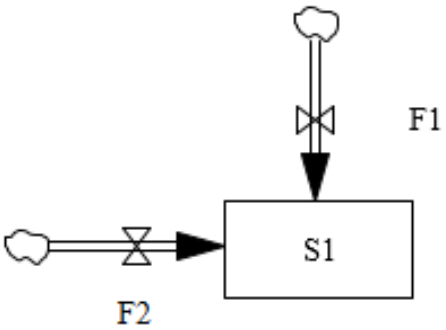

```
C/C++
int main(){
    Model m;
    System S1;
    MyFlow F1, F2;
    F1.setSource(&S1);
    F2.setSource(&S1);
    m.add(&S1);
    m.add(&F1);
    m.add(&F2);
    m.run(1, 100);
    return 0;
}
```

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



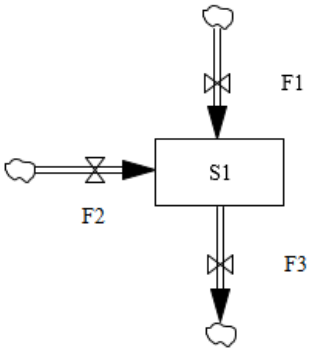
```C/C++
int main(){
    Model m;
    System S1;
    MyFlow F1, F2, F3;
    F1.setTarget(&S1);
    F2.setSource(&S1);
    F3.setSource(&S1);
    m.add(&S1);
    m.add(&F1);
    m.add(&F2);
    m.add(&F3);
    m.run(1, 100);
    return 0;
}
```

9 - Sistema recebendo múltiplos fluxos



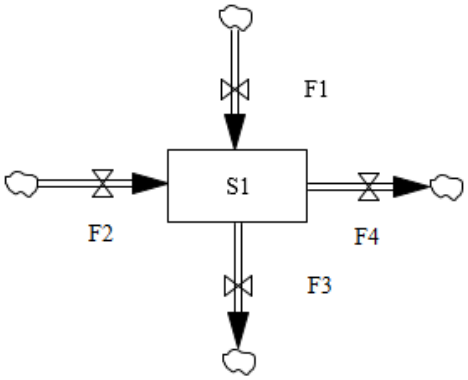

```C/C++
int main(){
    Model m;
    System S1;
    MyFlow F1, F2;
    F1.setTarget(&S1);
    F2.setTarget(&S1);
    m.add(&S1);
    m.add(&F1);
    m.add(&F2);
    m.run(1, 100);
    return 0;
}
```

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



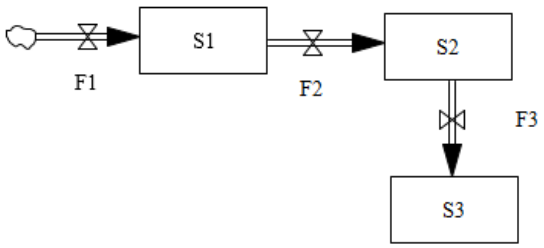

```C/C++
int main(){
    Model m;
    System S1;
    MyFlow F1, F2, F3;
    F1.setTarget(&S1);
    F2.setTarget(&S1);
    F3.setSource(&S1);
    m.add(&S1);
    m.add(&F1);
    m.add(&F2);
    m.add(&F3);
    m.run(1, 100);
    return 0;
}
```

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



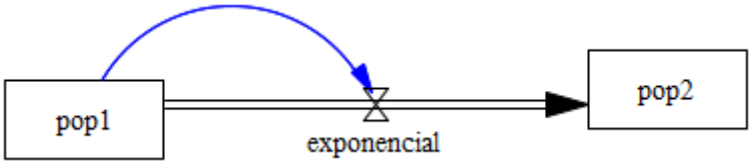
```C/C++
int main(){
    Model m;
    System S1;
    MyFlow F1, F2, F3, F4;
    F1.setTarget(&S1);
    F2.setTarget(&S1);
    F3.setSource(&S1);
    F4.setSource(&S1);
    m.add(&S1);
    m.add(&F1);
    m.add(&F2);
    m.add(&F3);
    m.add(&F4);
    m.run(1, 100);
    return 0;
}
```

12 - Múltiplos sistemas e fluxos




```C/C++
int main(){
    Model m;
    System S1, S2, S3;
    MyFlow F1, F2, F3;
    F1.setTarget(&S1);
    F2.setSource(&S1);
    F2.setTarget(&S2);
    F3.setSource(&S2);
    F3.setTarget(&S3);
    m.run(1, 100);
    return 0;
}
```

Critério de aceitação

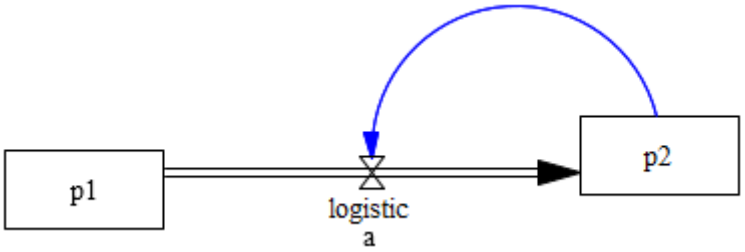


Um grande problema enfrentado durante a formulação dos modelos seria a de como atribuir uma função para o Flow, no exemplo seria a função exponencial que está conectando os sistemas pop1 e pop2.

Desta forma, pode-se utilizar uma função anônima que recebe um valor double de entrada e retorna, também, um valor double. Porém essa ideia acaba quebrando a programação orientada a objetos, já que qualquer função poderia ser válida.

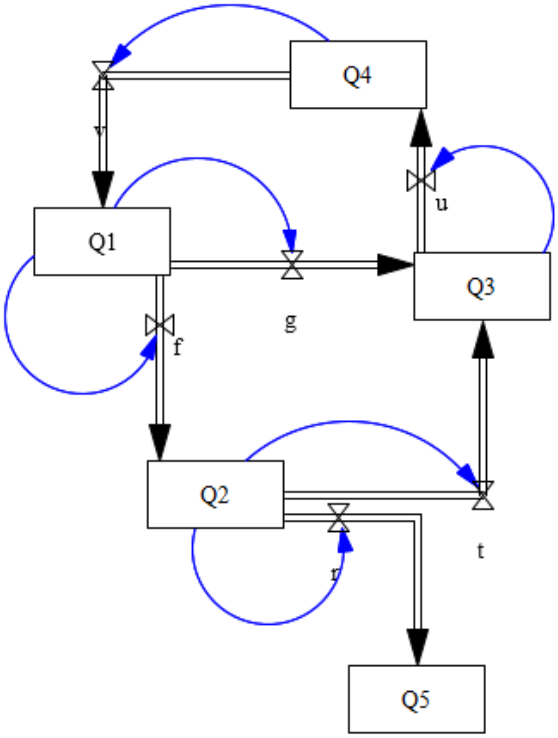
Então, outra solução é tornar a classe Flow abstrata, forçando o usuário a criar uma classe com a implementação de um método chamado de executeEquation().

```C/C++
class MyFlow : public Flow {
public:
    MyFlow() : Flow(){}
    double executeEquation() override {
        return 0.01 * this->getSource()->getValue();
    }
};
int main(){
    System * pop1 = new System("pop1", 100);
    System * pop2 = new System("pop2", 200);

    MyFlow * exponencial = new MyFlow();
    exponencial->setSource(pop1);
    exponencial->setTarget(pop2);

    Model m;
    m.add(pop1);
    m.add(pop2);
    m.add(exponencial);
    m.run(1, 100);
    return 0;
}
```



A função de logística seguirá o mesmo conceito da exponencial.

```C/C++
class MyFlow : public Flow {
public:
    MyFlow() : Flow(){}

    double executeEquation() override {
        double input = this->getSource()->getValue();
        return 0.01 * input * (1 - input / 70);

    }
};

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

    MyFlow * logistica = new MyFlow();
    logistica->setSource(p1);
    logistica->setTarget(p2);

    Model m;
    m.add(p1);
    m.add(p2);
    m.add(logistica);
    m.run(1, 100);

    return 0;
}
```

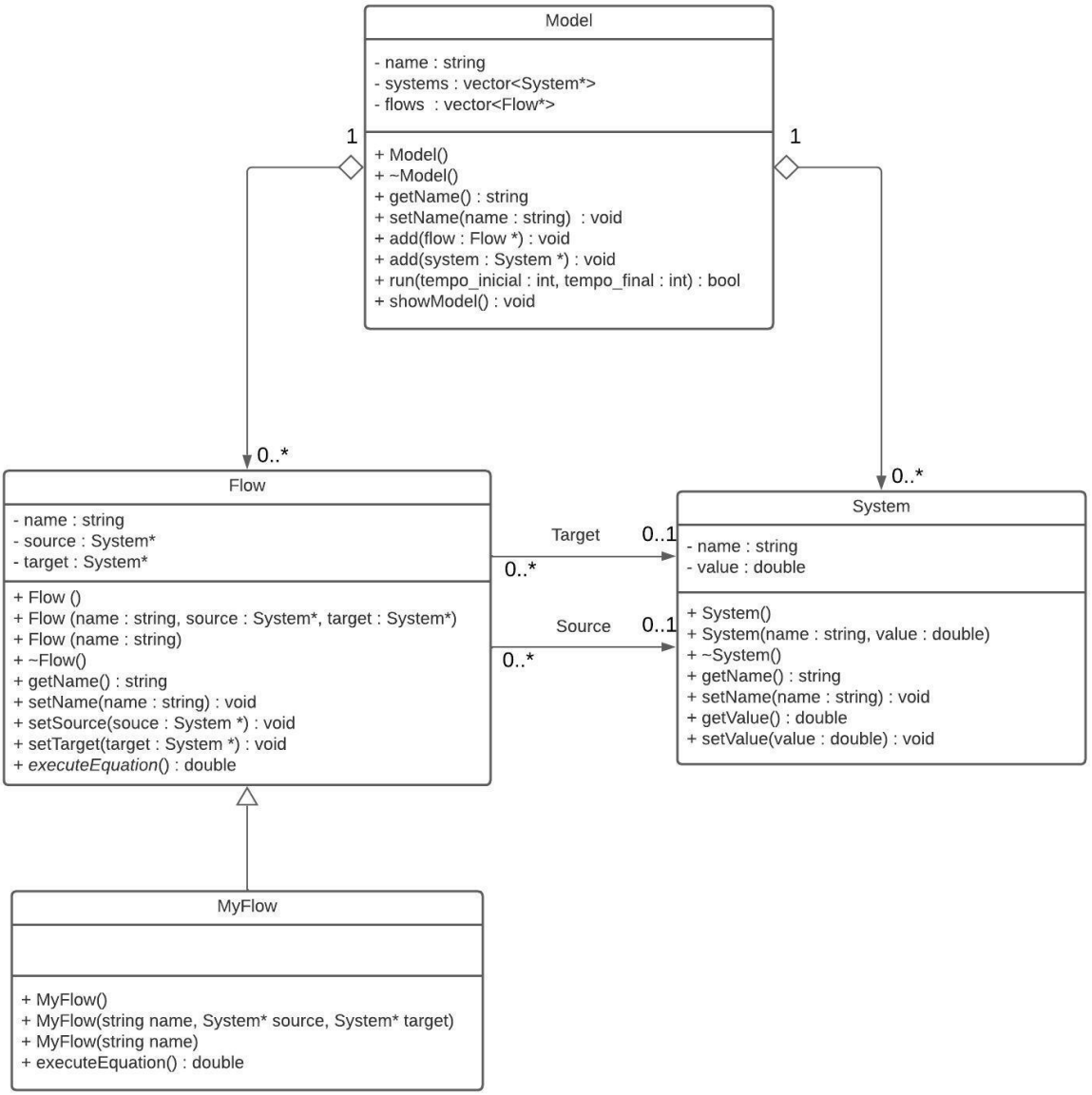


Uma sobrecarga no construtor do MyFlow pode ser realizada a fim de atribuir, diretamente na instanciação da classe, o nome, equação, origem e destino. Dessa forma, o código da API fica mais legível e enxuto.

```C/C++
class MyFlow : public Flow {
public:
    MyFlow(string name, System* source, System* target) : Flow(name, source,
target){}

    double executeEquation() override {
        return 0.01 * this->getSource()->getValue();
    }
};

int main(){
    System * Q1, * Q2, * Q3, * Q4, * Q5;

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

    m.run(1, 100);

    return 0;
}
```

UML


Model
- name : string
- systems : vector<System*>
- flows : vector<Flow*>
+ Model()
+ ~Model()
+ getName() : string
+ setName(name : string) : void
+ add(flow : Flow *) : void
+ add(system : System *) : void
+ run(tempo_inicial : int, tempo_final : int) : bool
+ showModel() : void
1
1
0..*
0..*
Flow
- name : string
- source : System*
- target : System*
+ Flow ()
+ Flow (name : string, source : System*, target : System*)
+ Flow (name : string)
+ ~Flow()
+ getName() : string
+ setName(name : string) : void
+ setSource(souce : System *) : void
+ setTarget(target : System *) : void
+ executeEquation() : double
Target
0..1
0..*
Source
0..1
0..*
System
- name : string
- value : double
+ System()
+ System(name : string, value : double)
+ ~System()
+ getName() : string
+ setName(name : string) : void
+ getValue() : double
+ setValue(value : double) : void
MyFlow
+ MyFlow()
+ MyFlow(string name, System* source, System* target)
+ MyFlow(string name)
+ executeEquation() : double