UNIVERSIDADE FEDERAL DE OURO PRETO
CAMPUS MORRO DO CRUZEIRO

Projeto da API do simulador de Sistemas Dinâmicos

Trabalho prático individual

Trabalho apresentado ao Professor Tiago Garcia de Senna Carneiro como parte das exigências da disciplina de Engenharia de Software I do curso de bacharelado em Ciência da Computação.

Alunos: Kézia Batista Alves da Conceição Brito

Ouro Preto

Outubro/2023

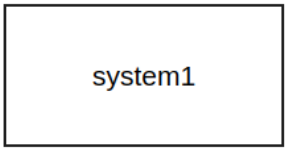
Sumário

1. Casos de uso

O primeiro passo consiste em estudar os casos de uso aos quais a API deve satisfazer. Com os casos de uso identificados, comecei a projetar a API, fazendo pseudo-códigos com base nos mesmos.

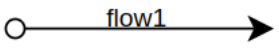
Não mudei de ideia quanto à estrutura da API durante esse desenvolvimento, pois os 4 casos apresentados em sala de aula pelo professor me deram uma base que julguei ser suficiente. Apenas achei conveniente a função *run* da classe *Model* receber, como parâmetro, a temporização de início e término da simulação, *startTime* e *endTime*, respectivamente.

1) Um sistema isolado.



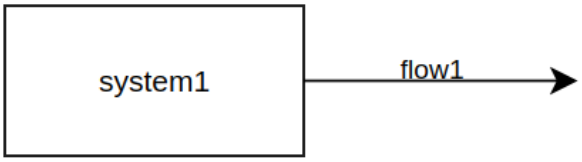
1. Model model;
2. System system1;
3. model.add(system1);
4. model.run(startTime, endTime);

2) Um fluxo isolado.



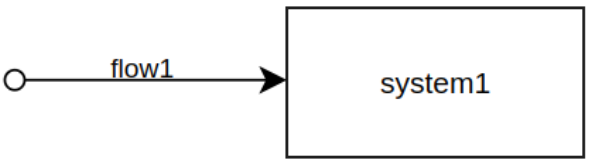
1. Model model;
2. Flow flow1;
3. model.add(flow1);
4. model.run(startTime, endTime);

3) Um sistema como origem de um fluxo.



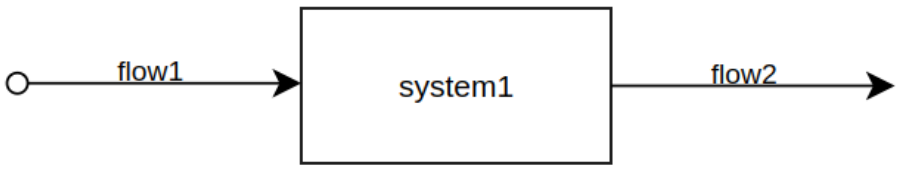
1. Model model;
2. System system1;
3. Flow flow1;
4. flow1.setSource(system1);
5. model.add(system1);
6. model.add(flow1);
7. model.run(startTime, endTime);

4) Um sistema como destino de um fluxo.



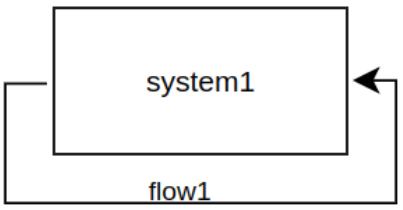
1. Model model;
2. System system1;
3. Flow flow1;
4. flow1.setTarget(system1);
5. model.add(system1);
6. model.add(flow1);
7. model.run(startTime, endTime);

5) Um sistema como destino de um fluxo e origem de outro.



1. Model model;
2. System system1;
3. Flow flow1, flow2;
4. flow1.setTarget(system1);
5. flow2.setSource(system1);
6. model.add(system1);
7. model.add(flow1);
8. model.add(flow2);
9. model.run(startTime, endTime);

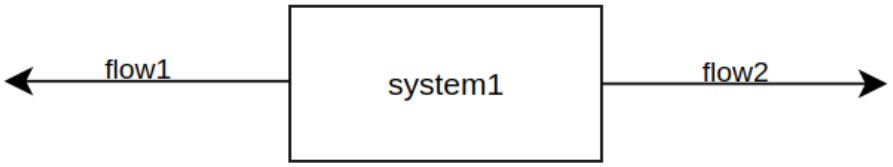
6) Um sistema como origem e destino de um fluxo.



1. Model model;
2. System system1;
3. Flow flow1;
4. flow1.setSource(system1);
5. flow1.setTarget(system1);

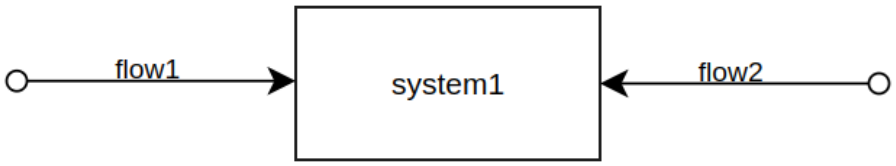
6. model.add(system1);
7. model.add(flow1);
8. model.run(startTime, endTime);

7) Um sistema como origem de vários fluxos, podendo, ou não, ter sistemas de destino.



1. Model model;
2. System system1;
3. Flow flow1, flow2;
4. flow1.setSource(system1);
5. flow2.setSource(system1);
6. model.add(system1);
7. model.add(flow1);
8. model.add(flow2);
9. model.run(startTime, endTime);

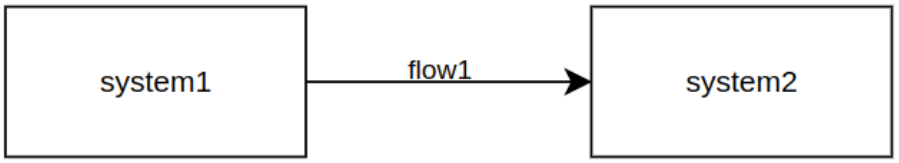
8) Um sistema como destino de vários fluxos, podendo, ou não, ter sistemas de origem.



1. Model model;
2. System system1;

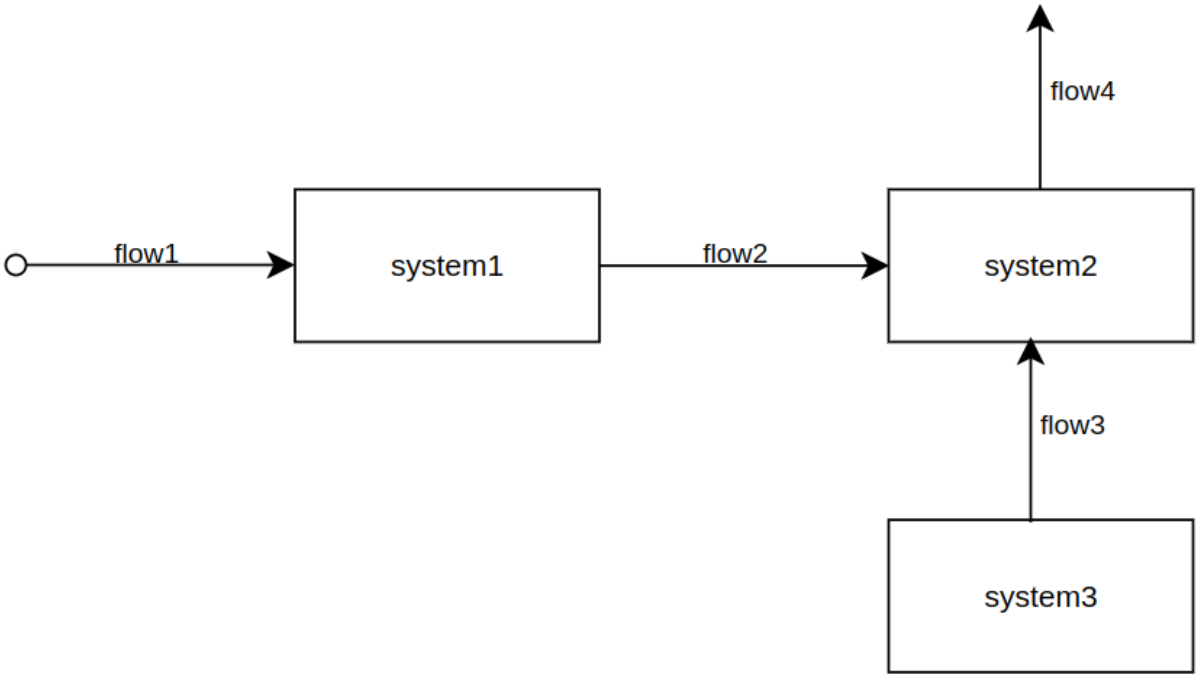
3. Flow flow1, flow2;
4. flow1.setTarget(system1);
5. flow2.setTarget(system1);
6. model.add(system1);
7. model.add(flow1);
8. model.add(flow2);
9. model.run(startTime, endTime);

9) Um fluxo com um sistema de origem e outro de destino.



1. Model model;
2. System system1, system2;
3. Flow flow1;
4. flow1.setSource(system1);
5. flow1.setTarget(system2);
6. model.add(system1);
7. model.add(system2);
8. model.add(flow1);
9. model.run(startTime, endTime);

10) Um conjunto de vários fluxos com sistemas de origem e de destino, podendo, ou não, ser cíclico.



1. Model model;
2. System system1, system2, system3;
3. Flow flow1, flow2, flow3, flow4;
4. flow1.setTarget(system1);
5. flow2.setSource(system1);
6. flow2.setTarget(system2);
7. flow3.setSource(system3);
8. flow3.setTarget(system2);
9. flow4.setSource(system2);
10. model.add(system1);
11. model.add(system2);
12. model.add(system3);
13. model.add(flow1);
14. model.add(flow2);
15. model.add(flow3);
16. model.add(flow4);

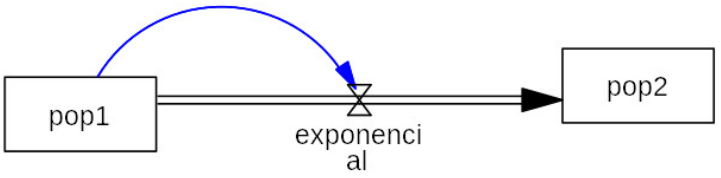
17. model.run(startTime, endTime);

Mesmo com os casos de uso concebidos e testados, ainda senti falta de algumas funcionalidades. Além de um modelo ser capaz de adicionar um sistema ou um fluxo, ele também deve poder **remover**, o que dá mais liberdade à simulação. Ainda pensando na liberdade da modelagem, o sistema e o fluxo devem poder ser **atualizados**.

É essencial, também, que **equações** possam ser atribuídas aos fluxos e estes possam **executá-las**. Inicialmente, pensei nas equações em formato de *string*, porém, isso requer análise léxica, sintática e semântica, tratando os erros em tempo de execução, o que deixaria muito mais lento e caro. Com isso, usar a herança para especializar o comportamento de uma classe é a solução mais adequada, ou seja, existe uma classe *Flow* abstrata com um método virtual puro *execute*, método este que, representando a equação, será adaptado pelo usuário-programador ao criar uma subclasse de *Flow*.

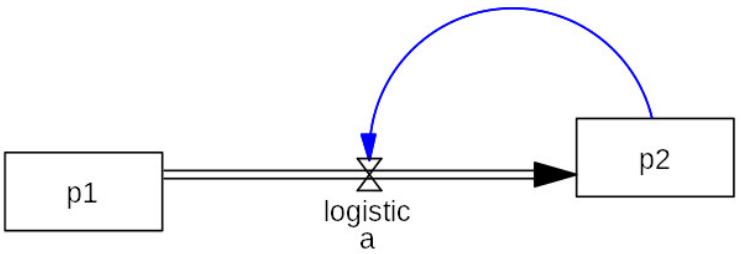
2. Critérios de aceitação

1)



1. Model model;
2. System pop1, pop2;
3. FlowExponencial flow1;
4. flow1.setSource(pop1);
5. flow1.setTarget(pop2);
6. model.add(pop1);
7. model.add(pop2);
8. model.add(flow1);
9. model.run(startTime, endTime);

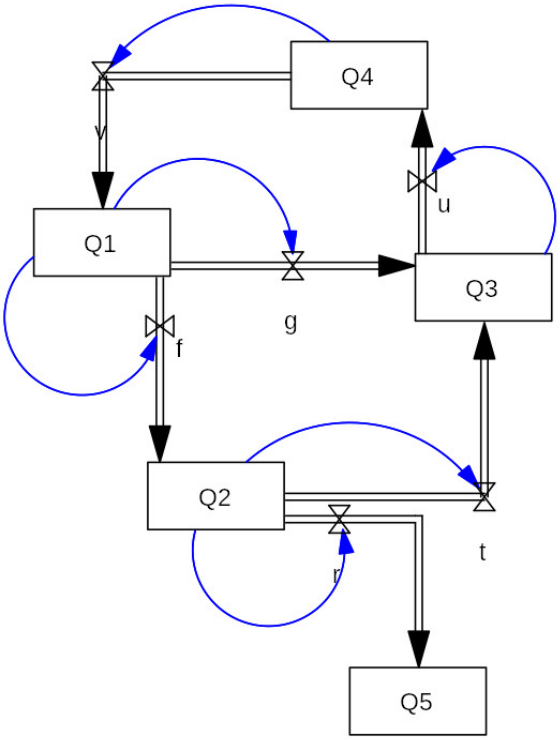
2)



1. Model model;
2. System p1, p2;
3. FlowLogistica flow1;
4. flow1.setSource(p1);
5. flow1.setTarget(p2);

6. model.add(p1);
7. model.add(p2);
8. model.add(flow1);
9. model.run(startTime, endTime);

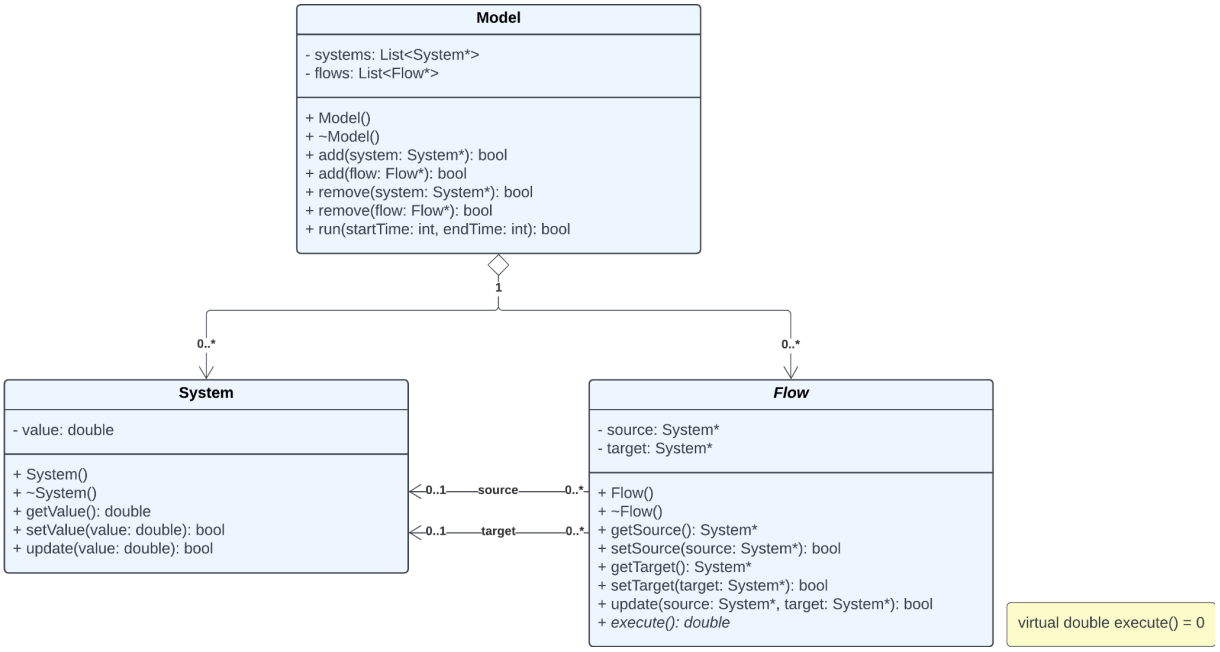
3)



1. Model model;
2. System Q1, Q2, Q3, Q4, Q5;
3. FlowF f;
4. FlowG g;
5. FlowR r;
6. FlowT t;
7. FlowU u;
8. FlowV v;
9. f.setSource(Q1);

```
10. f.setTarget(Q2);
11. g.setSource(Q1);
12. g.setTarget(Q3);
13. r.setSource(Q2);
14. r.setTarget(Q5);
15. t.setSource(Q2);
16. t.setTarget(Q3);
17. u.setSource(Q3);
18. u.setTarget(Q4);
19. v.setSource(Q4);
20. v.setTarget(Q1);
21. model.add(Q1);
22. model.add(Q2);
23. model.add(Q3);
24. model.add(Q4);
25. model.add(Q5);
26. model.add(f);
27. model.add(g);
28. model.add(r);
29. model.add(t);
30. model.add(u);
31. model.add(v);
32. model.run(startTime, endTime);
```

3. Diagrama UML



[Link para o diagrama.]