Universidade Federal de Ouro Preto - UFOP
Instituto de Ciências Exatas e Biológicas - ICEB
Departamento de Computação - DECOM
Ciência da Computação

Trabalho Individual - Sprint 1 - Concepção e Projeto da API em C++

BCC 322 - Engenharia de software

Lucas Chagas

Ouro Preto
24 de outubro de 2023

Decisões de Estruturas

Com os casos de uso e o que foi passado durante as aulas de engenharia de software, iremos investigar quais seriam as melhores decisões estruturais para o projeto da API.

Primeiramente, o objetivo era definir os “substantivos”(Classes) presentes no projeto, e com o auxílio do professor chegamos a conclusão de que seria necessário apenas 3 objetos para fazer um simulador com o tema teoria de sistemas, sendo esses objetos: Model, System e Flow.

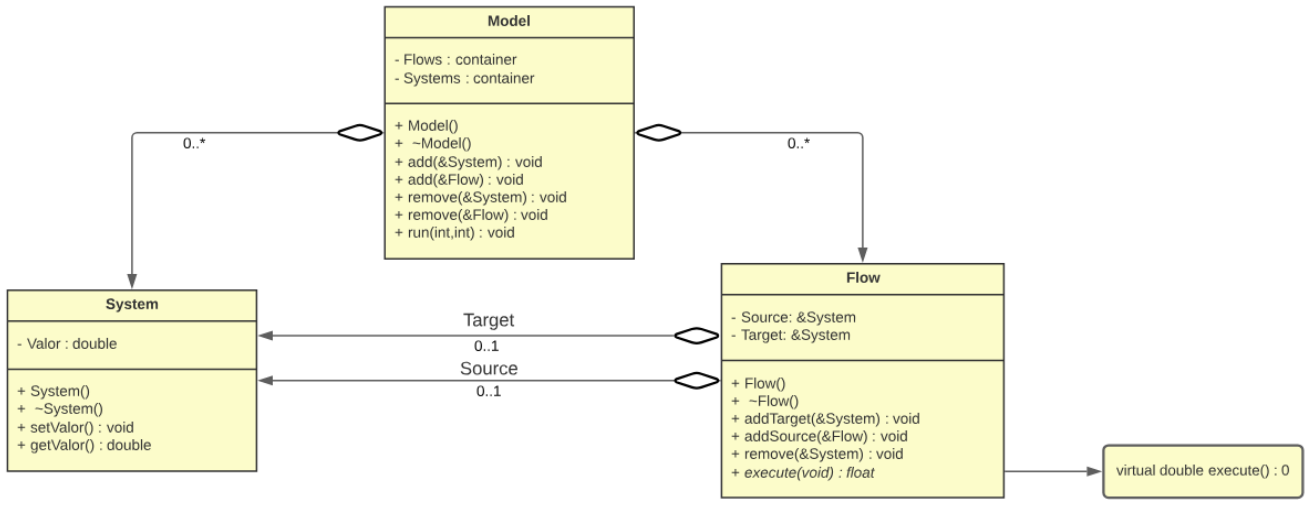
Após a definição dos “substantivos” o objetivo era definir os “verbos”(métodos), e foi realizado casos de uso a partir da decomposição dos critérios de aceitação fornecidos pelo professor para ter uma ideia mais sólida e funcional de quais métodos cada classe iria possuir. Tendo realizado os casos de uso, tanto os “substantivos” quanto os verbos foram definidos.

Classes

A Classe “Model” ficou responsável por armazenar todos os sistemas e fluxos através de métodos que adicionam sistemas ou fluxos para containers presentes na classe e por executar todo o funcionamento e interação entre sistemas e fluxos.

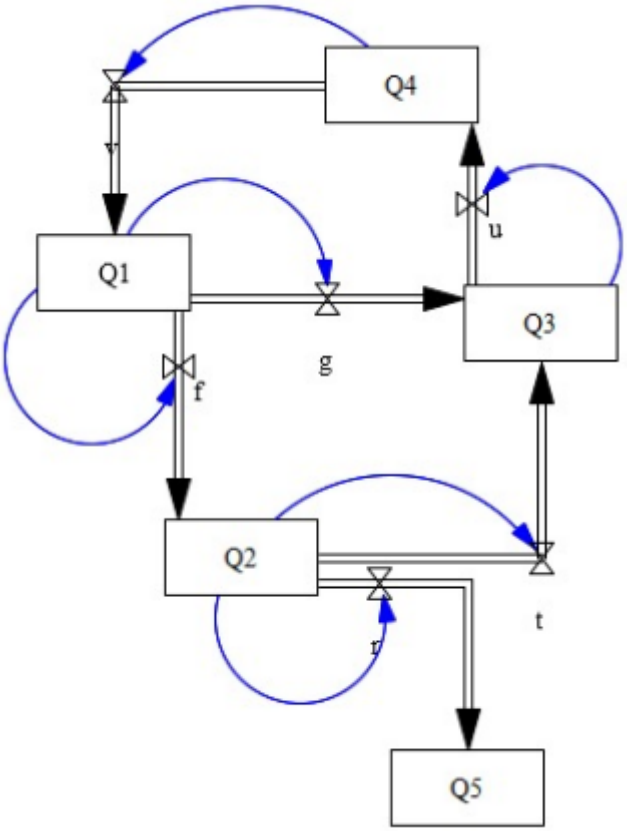
A Classe “System” possui apenas um valor que representa a “energia” que irá transitar entre os sistemas e fluxos.

A Classe “Flow” é responsável por realizar a conexão entre os sistemas, recebendo um sistema como origem e outro sistema como alvo. Outra função da classe é a definição de uma equação para realizar a alterações nos valores dos sistemas, porém, tendo em vista que o usuário deverá definir sua própria equação, a forma adotada para realizar tal tarefa foi tornar a classe abstrata para que o usuário possa posteriormente implementar sua própria equação por meio do método virtual “equation”.

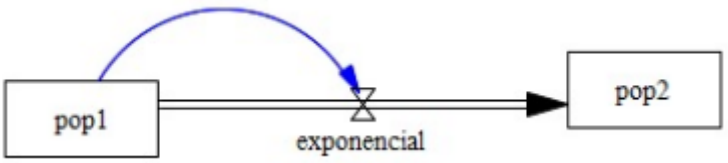


Casos de Uso

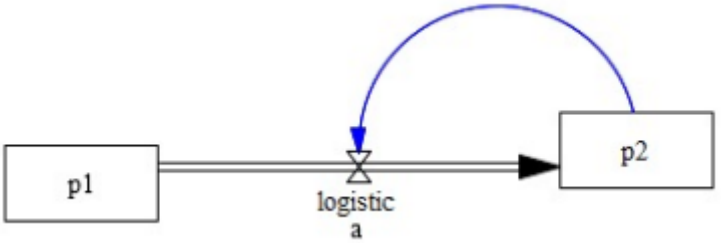
Os casos de uso foram feitos através de uma decomposição dos “Critérios de Aceitação” impostas, tendo como objetivo visualizar e simular o'que e quais são as funcionalidades necessárias para fazer com que o projeto possa passar pelos critérios de aceitação.



Critério de aceitação 1.



Critério de aceitação 2.



Critério de aceitação 3.

Caso 1:

Exemplo:

```C/C++
int main(){

	Model m1;
	System s1;

	m1.add(s1);
	m1.run(Tempoinicial,TempoFinal);

	return 0;
}
```

Caso 2:

Exemplo:

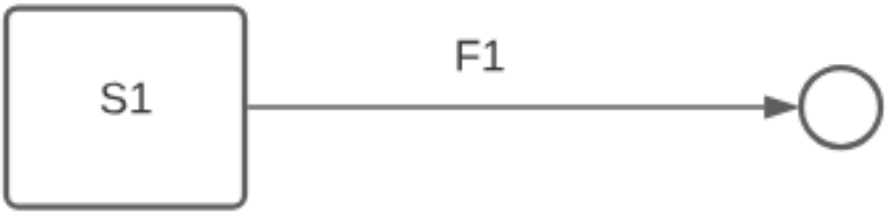
```C/C++
int main(){
	Model m1;
	Flow f1;
	m1.add(f1);
	m1.run(Tempoinicial,TempoFinal);
	return 0;
}
```

Caso 3:

Exemplo:

```
C/C++
int main(){
	Model m1;
	System s1;
	Flow f1;

	f1.addTarget(s1);
	m1.add(s1);
	m1.add(f1);

	m1.run(Tempoinicial,TempoFinal);
	return 0;
}
```

Caso 4:



Exemplo:

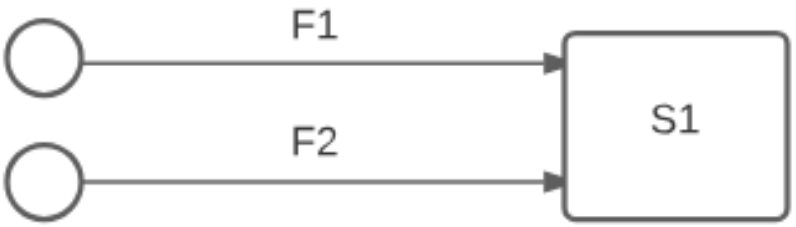
```C/C++
int main(){
	Model m1;
	System s1;
	Flow f1;

	f1.addTarget(s1);
	m1.add(s1);
	m1.add(f1);

	m1.run(Tempoinicial,TempoFinal);
	return 0;
}
```

Caso 5:

Exemplo:

```C/C++
int main(){
	Model m1;
	System s1,s2;
	Flow f1;
	f1.addTarget(s1);
	f1.addSource(s2);

	m1.add(s1);
	m1.add(s2);
	m1.add(f1);

	m1.run(Tempoinicial,TempoFinal);
	return 0;
}
```

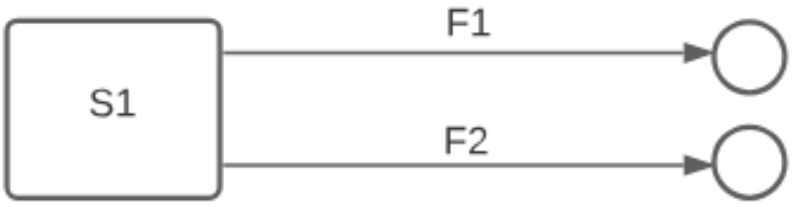
Caso 6:



Exemplo De código:

```C/C++
int main(){
	Model m1;
	System s1;
	Flow f1,f2;

	f1.addTarget(s1);
	f2.addTarget(s1);

	m1.add(s1);
	m1.add(f1);
	m1.add(f2);

	m1.run(Tempoinicial,TempoFinal);
	return 0;
}
```

Caso 7:



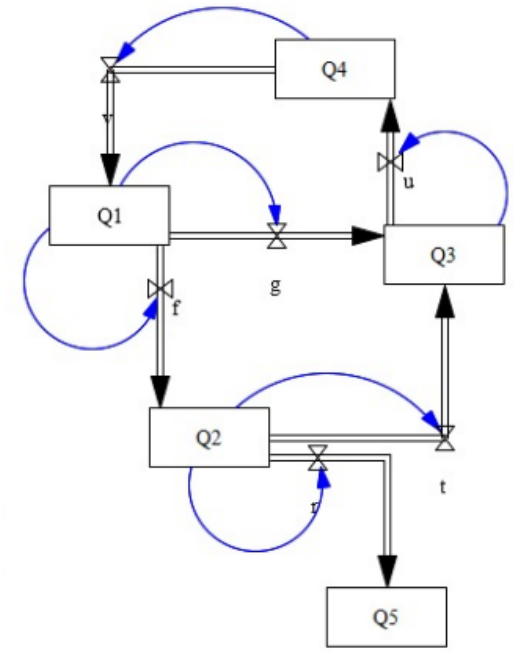
Exemplo De código:

```C/C++
int main(){
	Model m1;
	System s1;
	Flow f1,f2;

	f1.addSource(s1);
	f2.addSource(s1);
	m1.add(s1);
	m1.add(f1);
	m1.add(f2);

	m1.run(Tempoinicial,TempoFinal);
	return 0;
}
```

Critérios de Aceitação

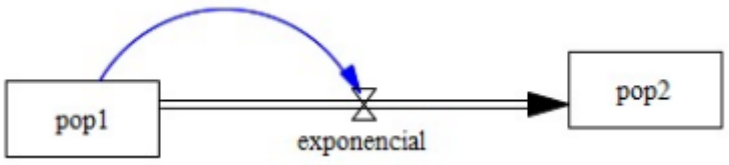
A partir dos casos de uso e respectivos códigos, será possível construir um código capaz de representar os critérios de aceitação.

Critério 1:



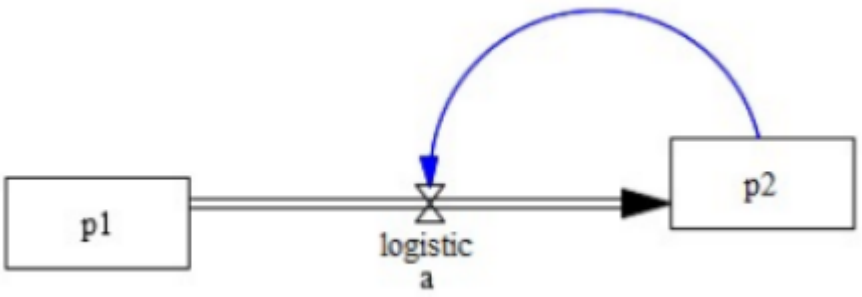
```C/C++
int main(){
	Model m1;
	System q1, q2, q3, q4, q5;
	Flow f, g, u, v, t, r;

	f.addSource(q1);
	f.addTarget(q2);
	m1.add(f);

	g.addSource(q1);
	g.addTarget(q3);
	m1.add(g);

	u.addSource(q3);
	u.addTarget(q4);
	m1.add(u);

	v.addSource(q4);
	v.addTarget(q1);
	m1.add(v);

	t.addSource(q2);
	t.addTarget(q3);
	m1.add(t);

	r.addSource(q2);
	r.addTarget(q5);
	m1.add(r);

	m1.add(q1);
	m1.add(q2);
	m1.add(q3);
	m1.add(q4);
	m1.add(q5);

	m1.run(Tempoinicial,TempoFinal);
	return 0;
}
```

Critério 2.



```C/C++
class Flow2 : public Flow{
	float equation(float x){
		return pow(x,2);
	}
}
int main(){
	Model m1;
	System pop1, pop2;
	Flow2 exponencial;

	exponencial.addTarget(pop2);
	exponencial.addSource(pop1);

	m1.add(exponencial);
	m1.add(pop1);
	m1.add(pop2);
	m1.run(Tempoinicial,TempoFinal)
	return 0;
}
```

Critério 3.




```C/C++
class Flow2 : public Flow{
	float equation(float x){
		return log(x);
	}
}
```

```
int main(){
	Model m1;
	System p1, p2;
	Flow2 logistica;

	logistica.addTarget(p2);
	logistica.addSource(p1);

	m1.add(logistica);
	m1.add(p1);
	m1.add(p2);
	m1.run(Tempoinicial,TempoFinal)
	return 0;
}
```