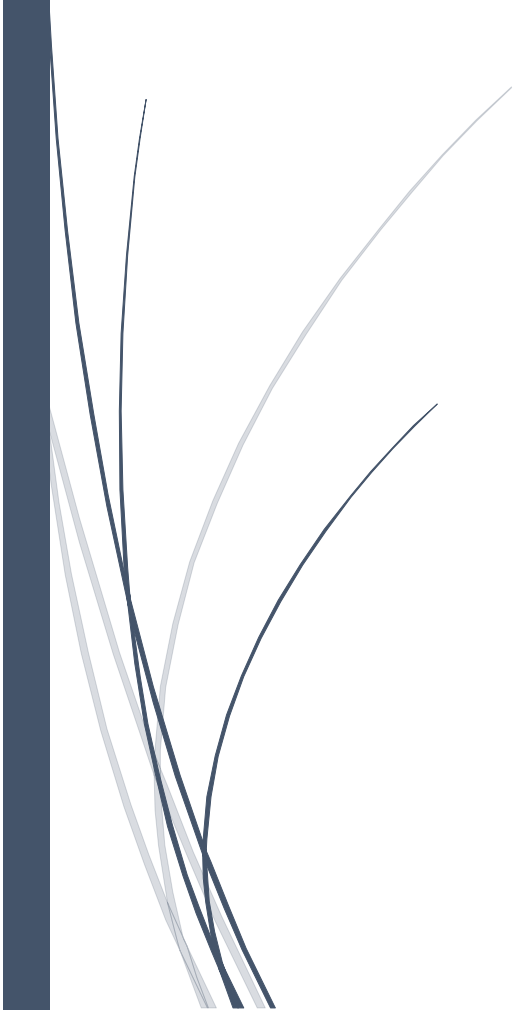
# JiStar

## Um Framework para descrever a intencionalidade do software



Ana Maria Moura

O framework JiStar oferece um conjunto de anotações de código Java para descrever a intencionalidade do software e uma ferramenta para exportar modelos de metas no formato HTML e/ou PiStar[i]. O framework JiStar baseia-se no framewok i* versão 1.0 [Eric 1995] para representar as informações de intencionalidade.

Antes do uso do framework JiStar, recomenda-se a leitura do guia de introdução ao framework i* em http://istar.rwth-aachen.de/tiki-index.php?page=iStarQuickGuide. Neste guia rápido, você encontrará a descrição dos elementos e relacionamentos presentes no i*.

## ANOTAÇÕES

A versão 1.0 do framework JiStar oferece as seguintes anotações de código Java.

- **@Actor(** *name**=”nome do ator” e *type**={ *general* | *agente* | *role* | *position*}**)**
  - Esta anotação pode ser utilizada em classe, interface (incluindo anotações), ou enumeração. Ela é utilizada para indicar cada ator que está sendo representado por determinado código. É possível utilizar mais de uma @Actor por unidade de código. Ao exportar um modelo i* em qualquer formato, somente os elementos associados a algum ator aparecerão no modelo.

```
@Actor(name = "Cozinheiro", type = ActorType.AGENT)
public class ReceitaGUI extends javax.swing.JFrame {...}
```

- **@Goal(** *name**=”nome da meta”, *description*=”descrição da meta” e *actor**= “nome do ator”**)**
  - Esta anotação pode ser utilizada em classe, interface (incluindo anotações) e enumeração para indicar as metas que os atores conseguem e/ou devem alcançar naquela unidade de código. É possível utilizar mais de uma @Goal por unidade de código.

```
@Actor(name = "Cozinheiro", type = ActorType.AGENT)
@Goal(name = "Que receitas sejam criadas", description = "Criar novas
receitas", actor=”Cozinheiro”)
public class ReceitaGUI extends javax.swing.JFrame {…}
```

- **@Softgoal(** *name**=”nome da meta flexível”, *description*=”descrição da meta flexível” e *actor**=”nome do ator”**)**
  - Esta anotação pode ser utilizada em classe, interface (incluindo anotações), enumeração e/ou atributo para relacionar as metas flexíveis (metas de qualidade) que os atores conseguem e/ou devem alcançar naquela unidade de código. É possível utilizar mais de uma @Softgoal por unidade de código. As operacionalizações destas metas flexíveis, quando disponíveis, serão indicadas pela anotação @Contribution.

```
@Actor(name="Cozinheiro", type=ActorType.AGENT)
...
@Softgoal(name = "Persistir as receitas", description = "Persistir
receita em base de dados", actor=”Cozinheiro”)
public class ReceitaGUI extends javax.swing.JFrame {…}
```

- **@Task(** *name**=”nome da tarefa” e *description**=”descrição da tarefa”**)**
  - Esta anotação pode ser utilizada em métodos para fornecer informações acerca da tarefa realizada pelo método em questão.

```
@Task(name="Gravar Receita", description="Grava a receita na base de
dados")
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt)
{…}
```

- **@Resource(** *name**=”nome do recurso”**)**
  - Esta anotação pode ser utilizada em tipos (i.e.: classe, interface (incluindo anotações), enumeração), atributo e/ou variável local para indicar recursos. É importante ressaltar que uma unidade de código como uma classe pode ser entendida como um recurso, como classes de entidade, por exemplo. No caso de uso em tipos, estes serão apresentados no modelo quando forem utilizados por algum ator e não poderão acumular a anotação @Actor.

```
@Resource(name="Receita")
public class Receita {
    private int id;
    private String nome;
    private String ingredientes;
    private String preparo;    ...}
```

- **@Contribution(** *type**=[*make* | *help* | *hurt* | *break*), *softgoal**=”meta flexível”**)**
  - Esta anotação pode ser utilizada em métodos para indicar um relacionamento de contribuição do método para uma meta flexível. Se o método possuir a anotação @Task, esta seja relacionada a meta flexível em questão. Caso contrário, será criada uma tarefa com o nome do método. É possível utilizar mais de uma @Contribution por unidade de código.

```
@Contribution(type=ContributionType.MAKE, softgoal="Persistir as
receitas")
...
 private void jButton2ActionPerformed(java.awt.event.ActionEvent evt)
{
        try{
        ReceitaControle controle = new ReceitaControle();
        Receita r = new Receita(txtNome.getText()); ...}
```

- **@TaskDeomposition(**type***=**[resource_for | softgoal_for | sub_goal | sub_task], elemento*=”nome do elemento”**)**
  - Esta anotação pode ser utilizada em métodos, uso de algum tipo ou declaração de parâmetro. Ela indica um relacionamento de decomposição de uma tarefa para uma subtarefa, um recurso para a tarefa, uma submeta ou uma meta flexível para a tarefa. É possível utilizar mais de uma @TaskDecomposition por unidade de código.

```
@TaskDecomposition(type=TaskDecompositionType.RESOURCE_FOR,
element="Receita")
...
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
...
        try{
        ReceitaControle controle = new ReceitaControle();
        @Resource(name = "Receita") Receita r = new
Receita(txtNome.getText()); ...}
```

- **@MeansEnd(** *endType**=[resource | goal], *end**=”nome da meta ou recurso”**)**
  - Esta anotação pode ser utilizada em métodos para indicar que aquele método é uma tarefa (meio) para alcançar determinada meta, meta flexível ou recurso. É possível utilizar mais de uma @MeansEnd por unidade de código.

```
@MeansEnd(endType=MeansEndType.GOAL,end="Criar receitas")
...
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
...
        try{
        ReceitaControle controle = new ReceitaControle();
        @Resource(name = "Receita") Receita r = new
Receita(txtNome.getText());   ...}
```

O framework JiStar permite a exportação das metainformações referente a intencionalidade do código para modelos de metas em i* no formato HTML e/ou PiStar. Um modelo de metas facilita a leitura das metas por parte dos engenheiros de requisitos, clientes e todas as partes interessadas no negócio suportado pelo sistema. Com o uso do framework JiStar é possível manter uma rastreabilidade entre o código e as metas do sistema, possibilitando ter um modelo de metas atualizado em mãos, onde todas as partes interessadas podem contribuir para a evolução do software.

Para gerar o modelo de metas a partir do código de fonte anotado, após engenheiro de software anotar o código, deve-se executar o gerador de modelo de metas passando o caminho dos arquivos compilados do projeto (*.class) e o formato de modelo desejado (-html ou -pistar), conforme trecho de comando abaixo (exemplo para Windows). O JiStar gera um arquivo chamado goal_model que pode ser carregado na ferramenta PiStar ou gera páginas html para cada aor do sistema.

```
java -jar JiStar-1.0.jar "caminho dos arquivos .class" -pistar
```

## Referências

Eric, S. Y. (1995). Modelling strategic relationships for process reengineering.

[i] https://www.cin.ufpe.br/~jhcp/pistar/#