Métodos de Desenvolvimento de Software (MDS) 2016/2017

Modelo de actividade

2

Diagramas de Actividade

Introdução

A partir do UML 2.0 a sua semântica é baseada em PetriNets (claramente distinta dos statecharts)

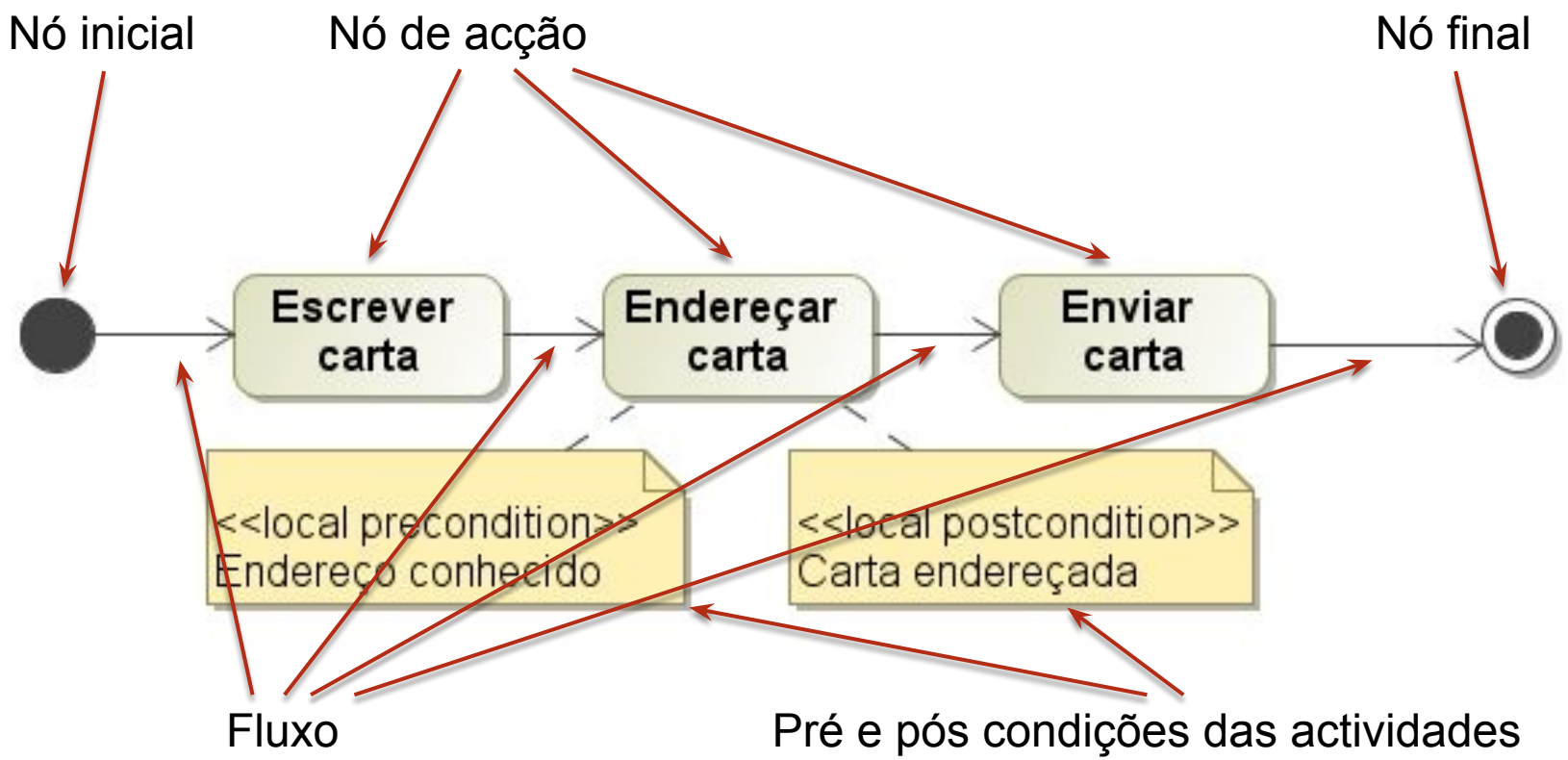
Diagramas de actividade

- Modelam aspectos dinâmicos de um sistema.
- Também modelam o fluxo de controlo de uma operação, classe, sistema, subsistema.
- Podem ser utilizados para descrever cenários de casos de uso.
- Enfatizam o fluxo de controlo através das mensagens entre objectos.
- Descrevem um processo consistindo em:
  - acções e actividades;
  - fluxo de controlo;
  - objectos de entrada e saída;
  - decisões;
  - concorrência,
  - ...

Actividades

- Actividades são representadas como redes de **nós** ligados por **arestas**
- Categorias de **nós**:
  - Nós de acção
    - Unidades discretas de trabalho, atómicas no contexto da actividade
  - Nós de controlo
    - Controlam o fluxo na actividade
  - Nós de objecto
    - Representam os objectos na actividade
- Categorias de arestas
  - Fluxo de controlo
    - Representa o fluxo de controlo na actividade
  - Fluxo de objectos
    - Representa o fluxo de objectos na actividade

Exemplo de diagrama de actividades

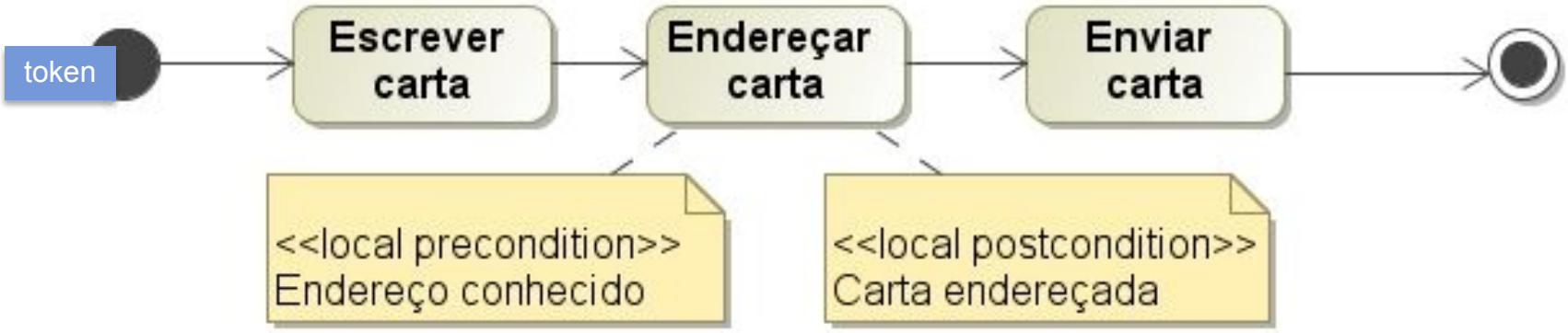


Também podemos definir pré e pós condições para a actividade.

Semântica das actividades

- Tokens:
  - Representam o controlo, objectos, ou dados
  - Deslocam-se do nó de origem para o nó de destino, através da aresta de fluxo
  - Apenas se deslocam quando **TODAS** as condições são satisfeitas

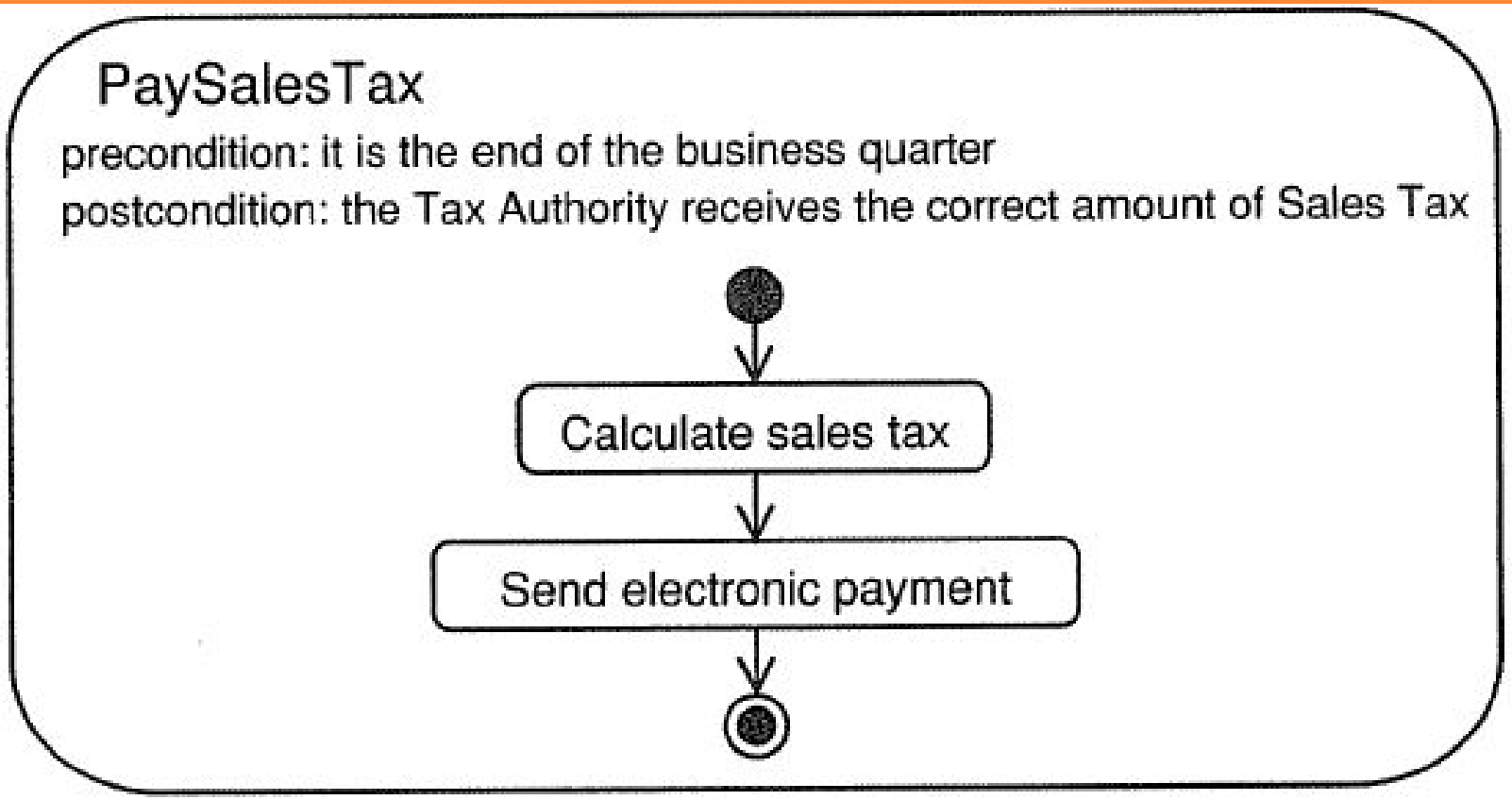
Mais sobre estes, dentro de momentos



Representação de cenários de Use Cases

Use case: PaySalesTax
ID: 1
Brief description: Pay Sales Tax to the Tax Authority at the end of the business quarter.
Primary actors: Time
Secondary actors: TaxAuthority
Preconditions: 1. It is the end of the business quarter.
Main flow: 1. The use case starts when it is the end of the business quarter. 2. The system determines the amount of Sales Tax owed to the Tax Authority. 3. The system sends an electronic payment to the Tax Authority.
Postconditions: 1. The Tax Authority receives the correct amount of Sales Tax.
Alternative flows: None.

Representação de cenários de Use Cases



Conceitos fundamentais

Conceitos

- Uma actividade corresponde à execução de um conjunto de acções.
- Uma acção é uma "computação" atómica, no contexto da actividade.
- Um diagrama de actividade contém:
  - Nós de acção
  - Nós de actividade
  - Transições
  - Objectos
  - Decisões (*branching*)
  - Disjunção (*fork*) e junção (*join*)
  - Pistas (*swimlanes*)

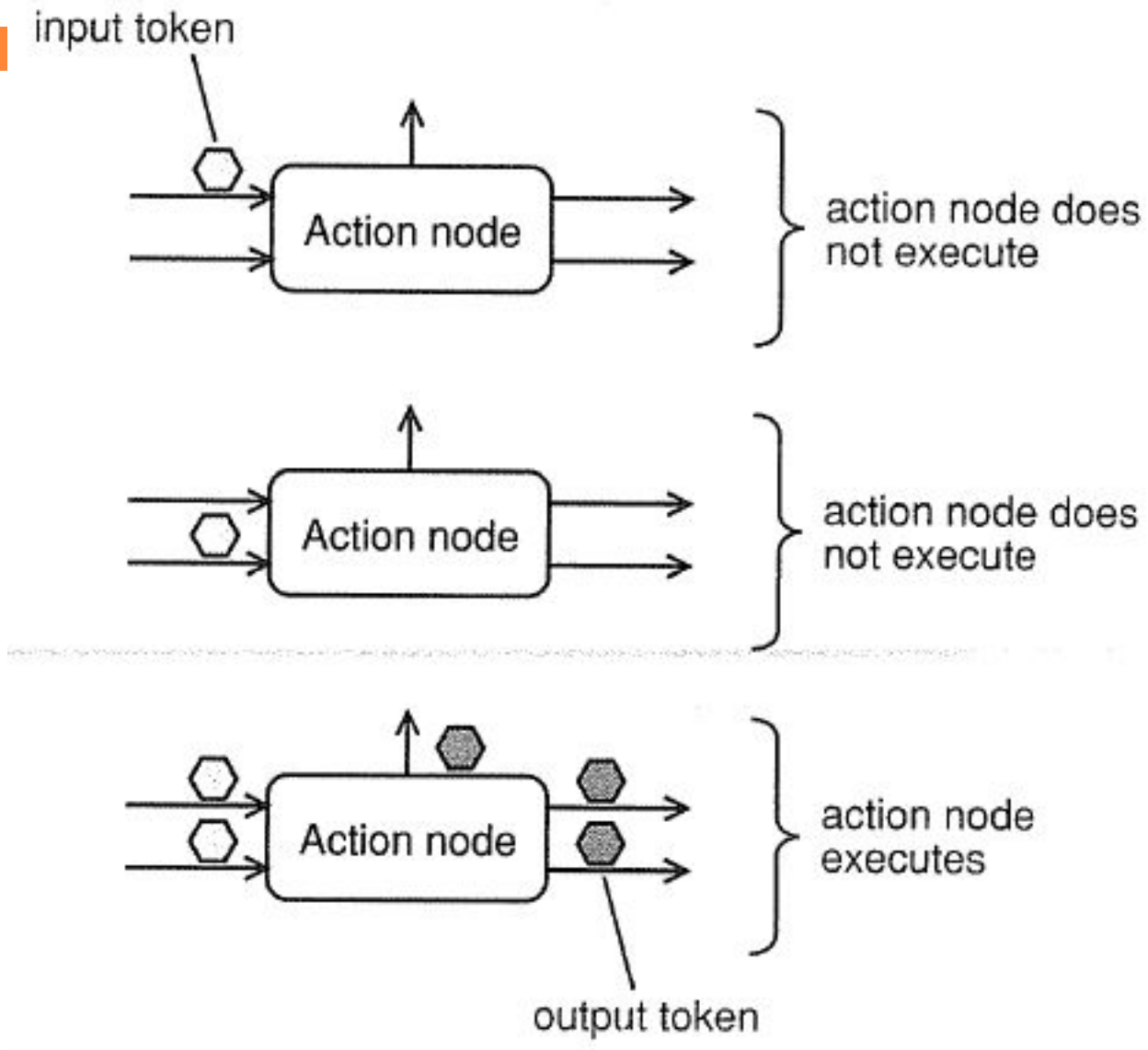
Nós de acção

- São nós do sistema representando a execução de uma acção:
  - e.g., criar ou destruir um objecto, executar uma operação num objecto
- São atómicos:
  - Não podem ser decompostos
  - Não podem ser interrompidos.
- O tempo de execução de um nó de acção é considerado insignificante.

Calculate total

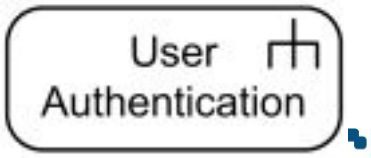
Create Account

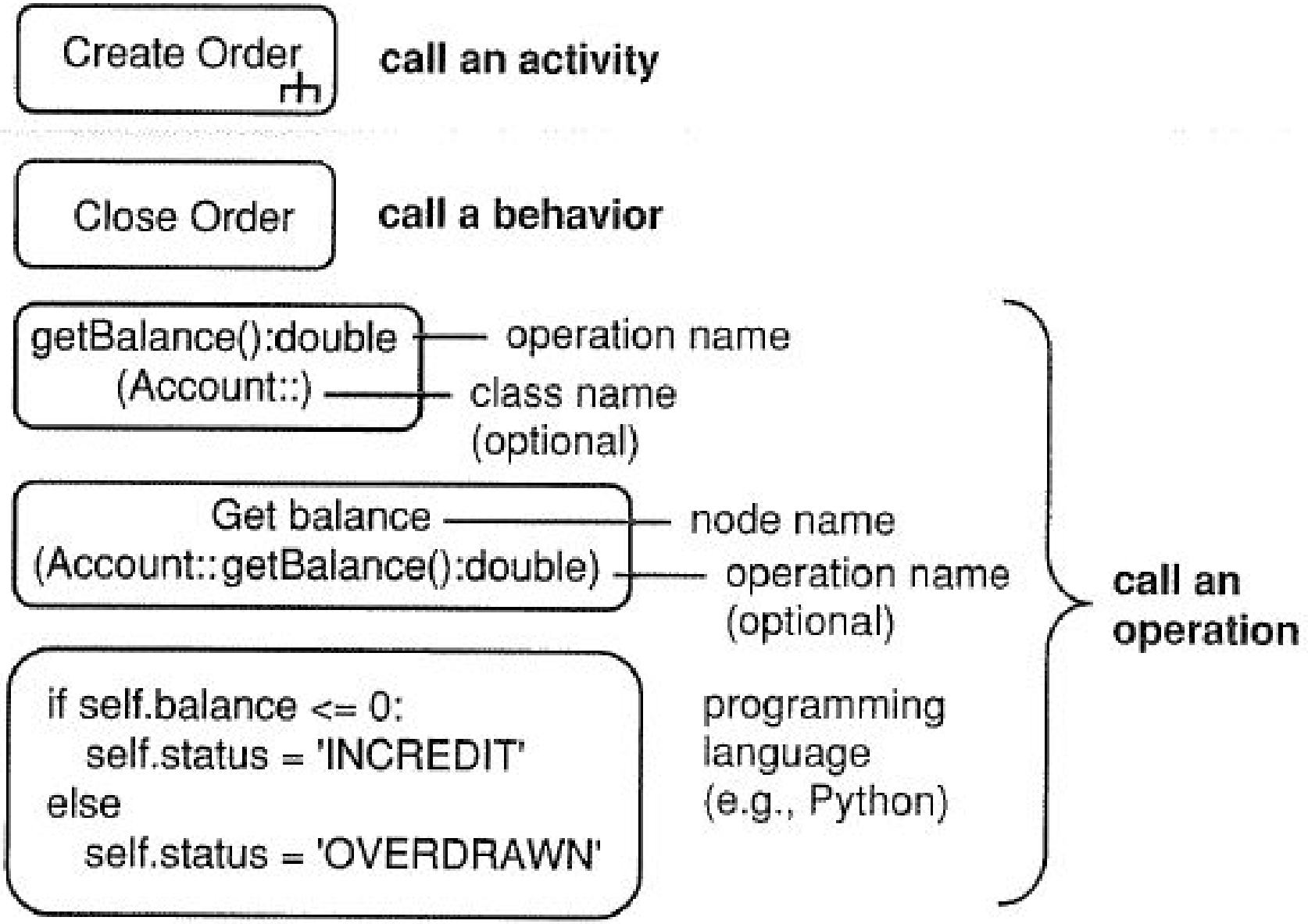
Nós de acção


input token
Action node
action node does not execute
Action node
action node does not execute
Action node
action node executes
output token

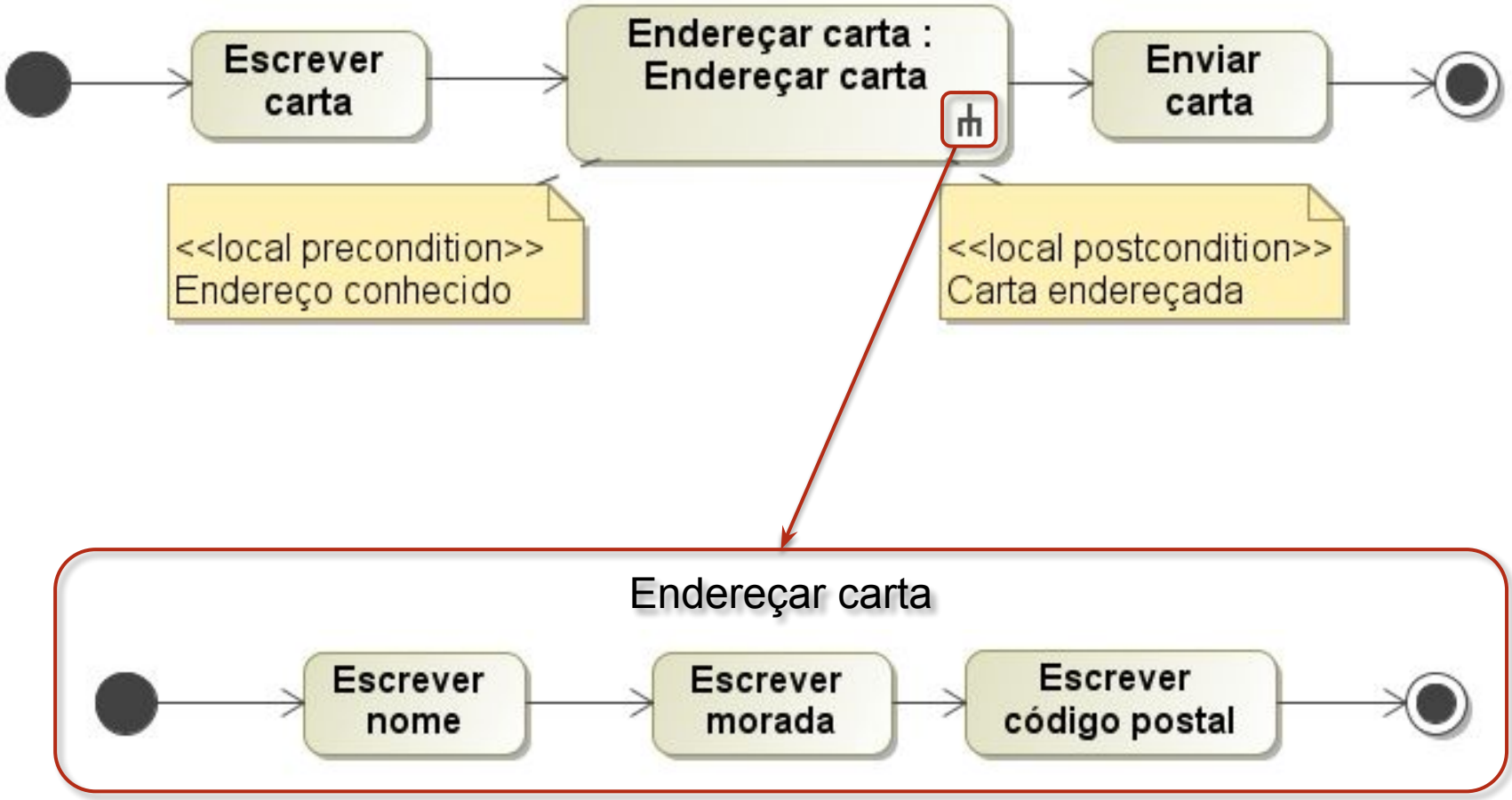
Nós de actividade

- Podem ser decompostos; onde se usa uma actividade pode usar-se outro diagrama de actividades.
- Não sendo atómicos, podem ser interrompidos (têm tempo (significativo) de execução).
- Um estado de acção é um nó de actividade que não pode ser decomposto.
- A notação é semelhante à do nó de acção usando-se uma anotação especial para indicar a existência do respectivo diagrama de actividade

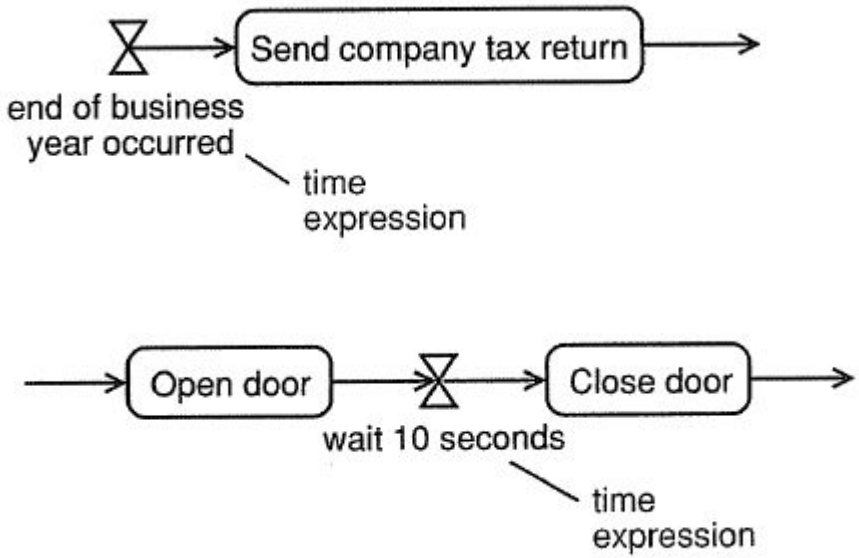



Create Order
call an activity
Close Order
call a behavior
getBalance():double
(Account::)
operation name
class name
(optional)
Get balance
(Account::getBalance():double)
node name
operation name
(optional)
call an
operation
if self.balance <= 0:
self.status = 'INCREDIT'
else
self.status = 'OVERDRAWN'
programming
language
(e.g., Python)

Estados de actividade



Nó de evento temporal



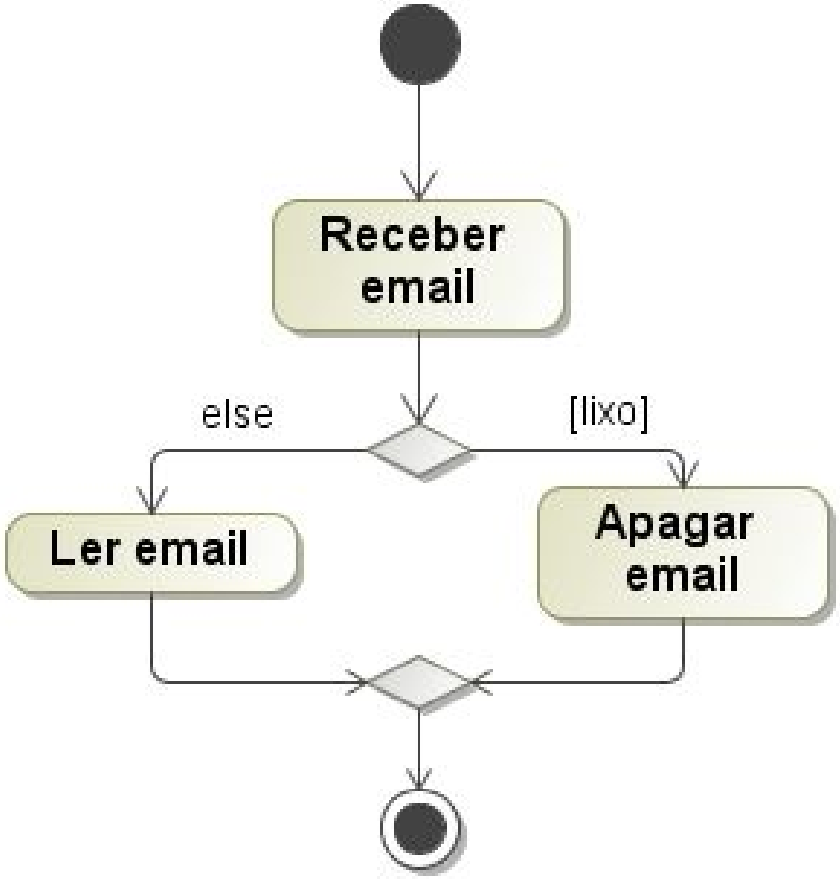
Expressão temporal pode ser:

- um evento no tempo (ex. último dia de negócios do ano)
- uma data específica (ex. 15h de 11/2/2017)
- uma duração (ex. esperar 15 segundos)

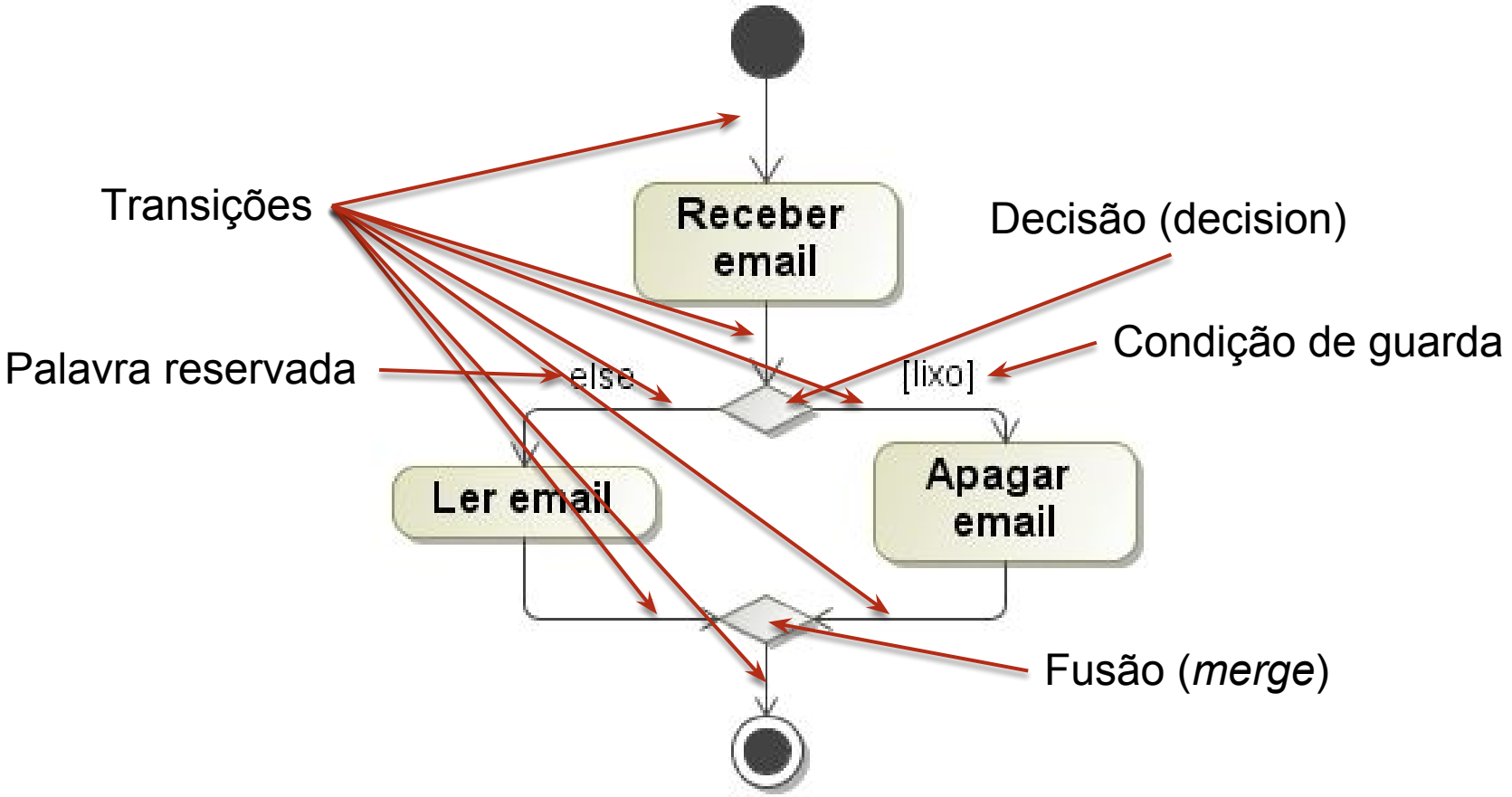
Transições e decisões

- **Transições**: quando a acção ou actividade de um estado se completa, o fluxo de controlo passa imediatamente para o próximo estado de acção ou actividade.
- **Decisões** (*branching*): expressões booleanas que especificam passos alternativos.
  - Um decisão consiste numa transição de entrada e duas ou mais de saída.
- **Guardas**: condições.
  - Nas transições de saída as guardas não se devem sobrepor, mas devem cobrir todas as possibilidades.

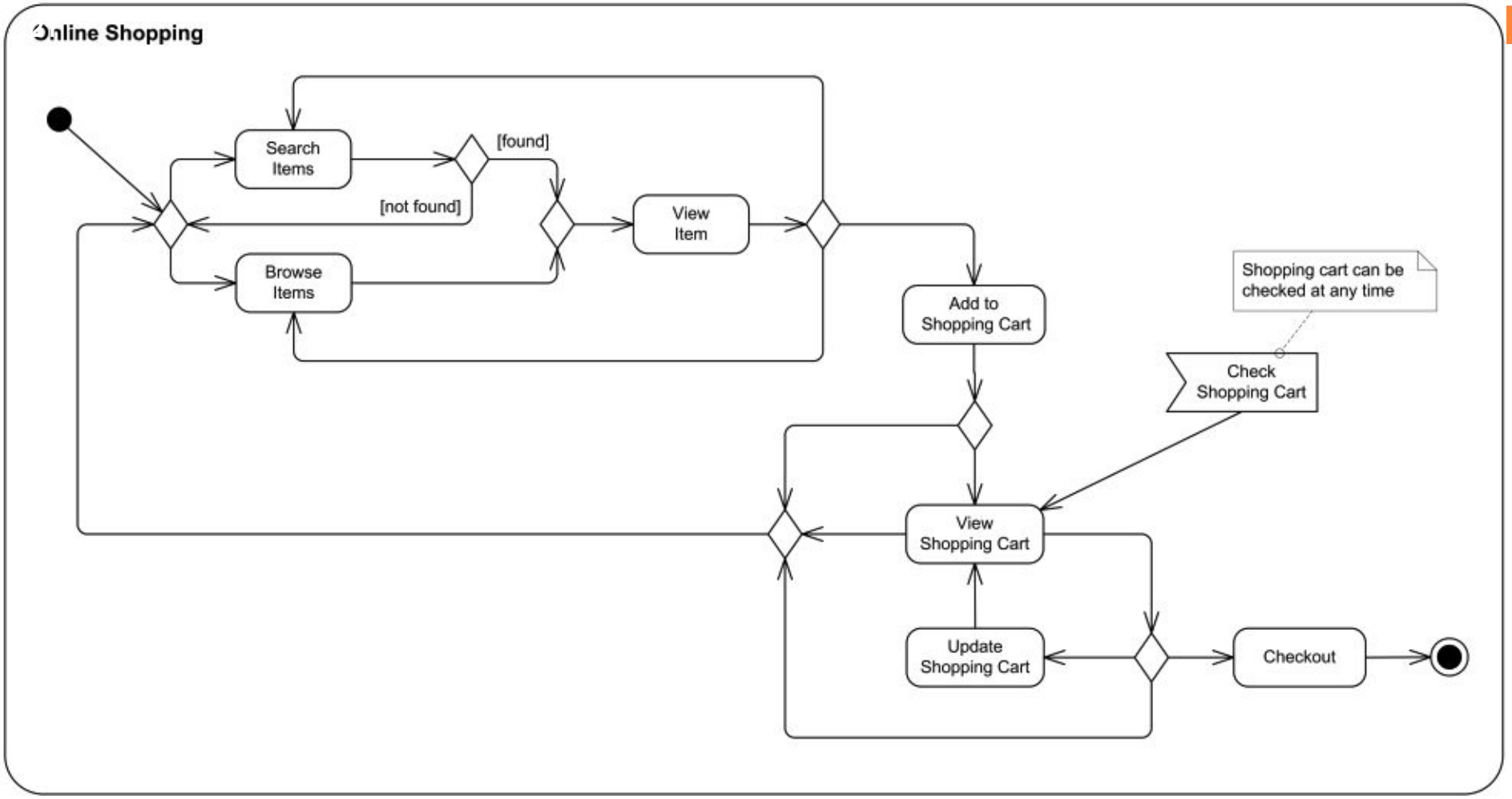
Transições e decisões



Transições e decisões

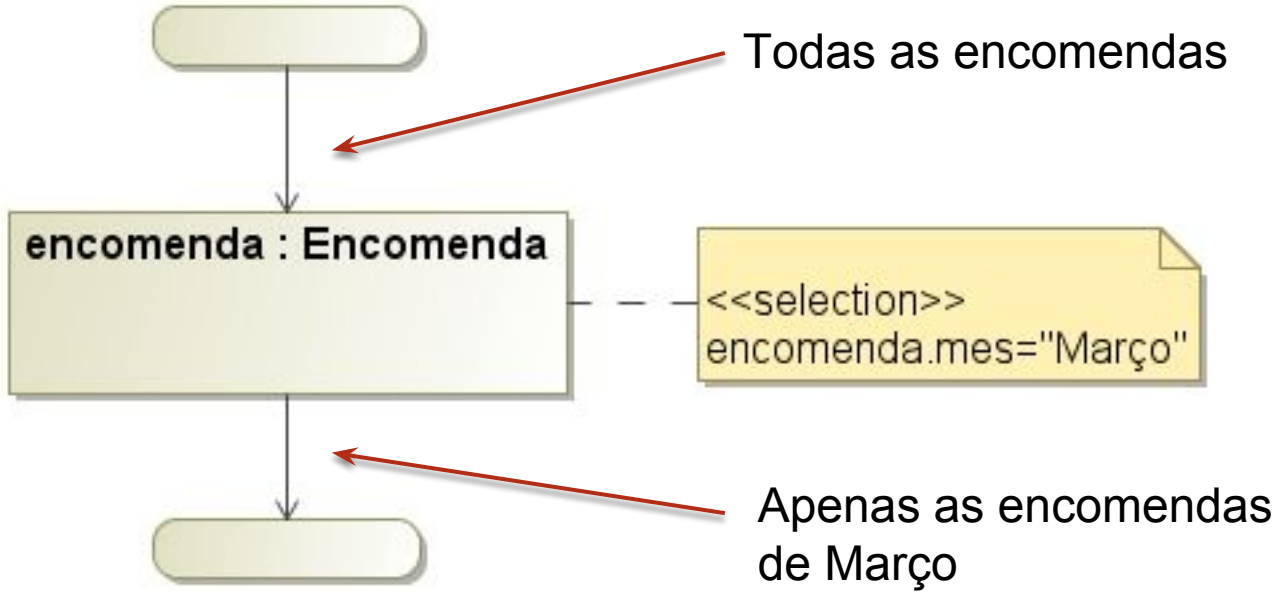


Decisões e fusões (*Decision* and *Merge*)



[http://www.uml-diagrams.org/notation/activity-edge-connector.png]

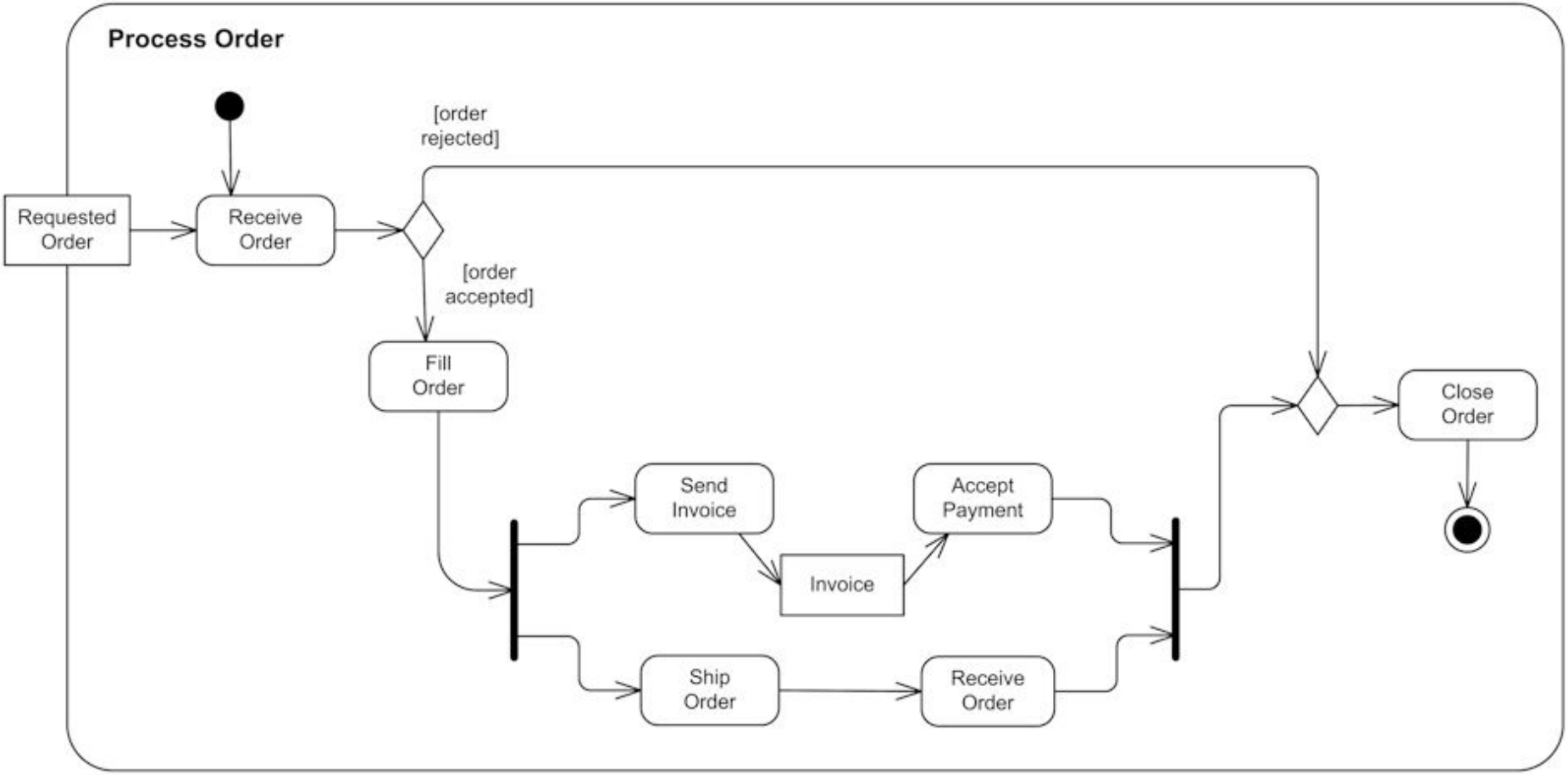
Nós de objecto



Disjunção (fork) e junção (join)

- Uma **disjunção/difusão** representa a separação de um fluxo de controlo em dois ou mais fluxos de controlo.
  - Pode ter uma transição de entrada e duas ou mais transições de saída.
- Uma **junção** representa a sincronização de dois ou mais fluxos de controlo.
  - Pode ter duas ou mais transições de entrada e uma de saída;
  - Os fluxos concorrentes sincronizam-se assim: espera-se que todos os fluxos de entrada cheguem ao ponto de junção prosseguindo com apenas um fluxo depois da junção.

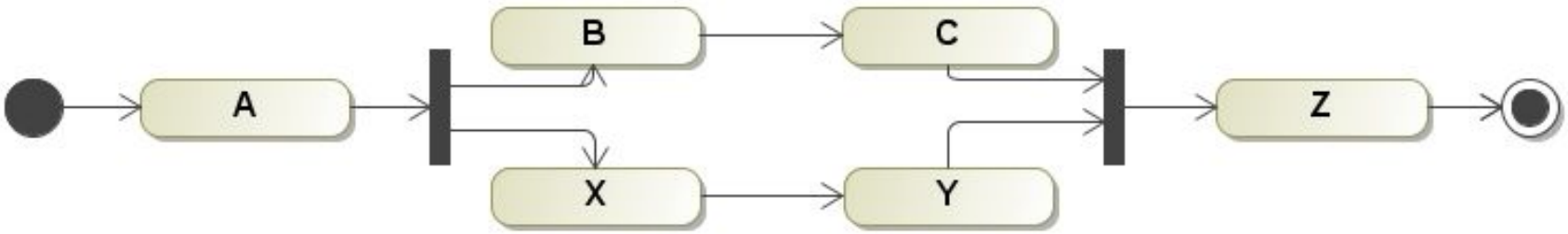
Exemplo de *fork* e *join*



[http://www.uml-diagrams.org/notation/activity-edge-connector.png]

Modelo de concorrência

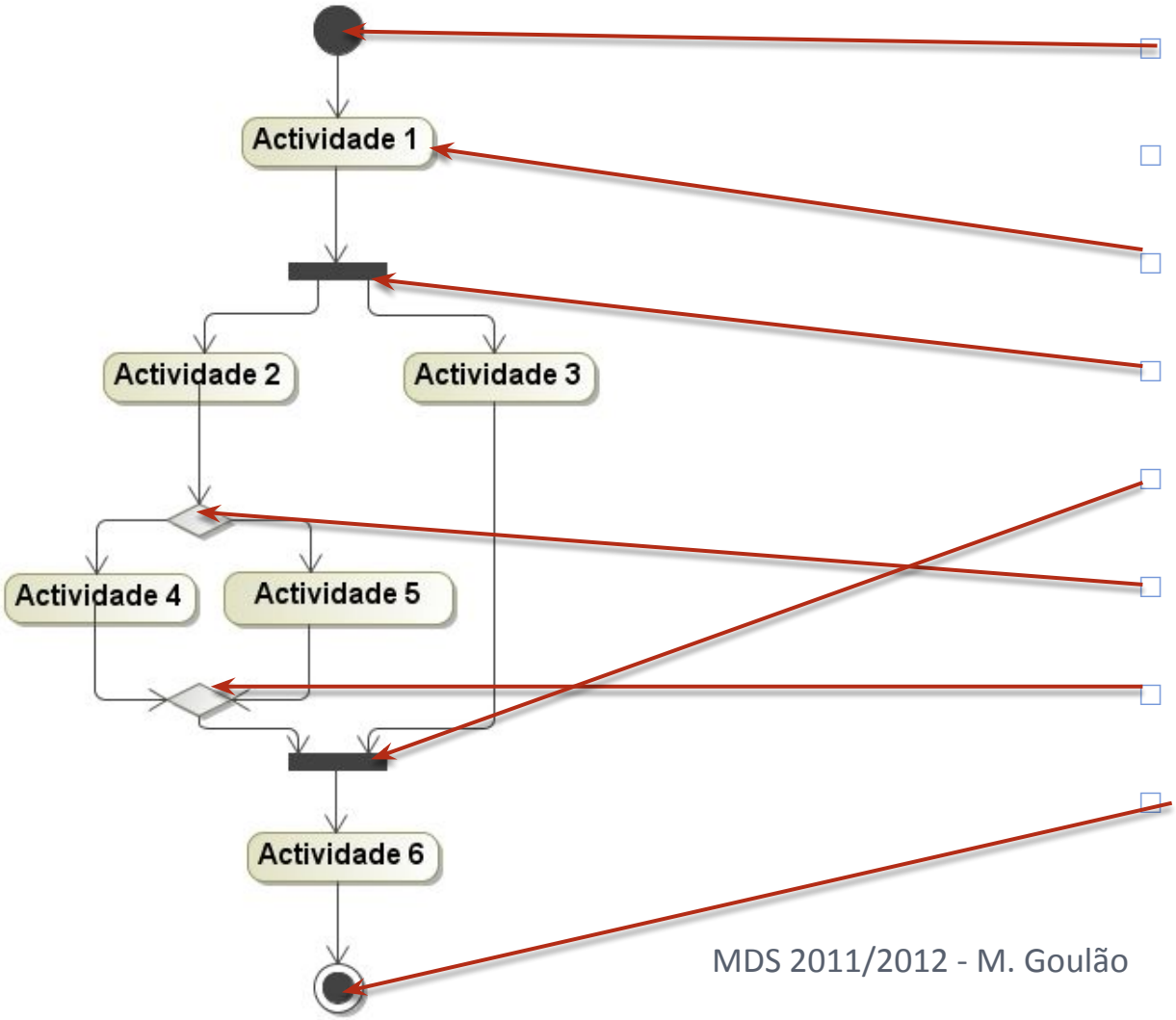
- *Fully independent concurrent streams*



Trace: A, {(B,C) || (X,Y)}, Z

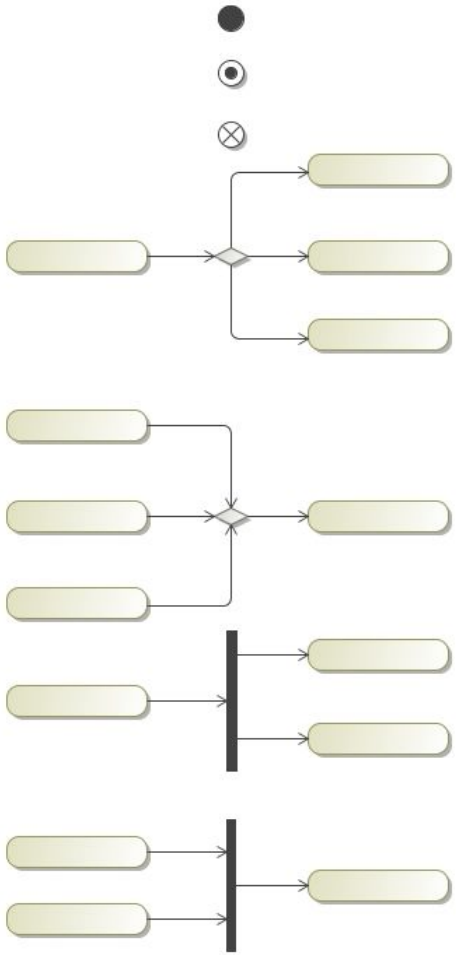
- Onde "||" é o operador de concorrência (usado nas linguagens de especificação algébricas, por exemplo)

Resumo



- Início
- Acções
- Actividades
- Fork
- Join
- Branch
- Merge
- Fim

Nós de controlo (Resumo)



- Nó inicial
  - Indica onde começa o fluxo
- Nó final
  - Termina uma actividade
- Nó final de fluxo
  - Termina um fluxo, sem afectar os restantes fluxos da actividade
- Nó de decisão
  - O fluxo de saída cuja guarda é verdadeira é atravessado
  - Pode ter, opcionalmente, um <<decisionInput>> que permita escolher o fluxo de saída adequado
- Nó de fusão
  - Copia os tokens de input para o fluxo de output
- Nó de disjunção (fork)
  - Separa o fluxo de entrada em vários fluxos de saída concorrentes
- Nó de junção (join)
  - Sincroniza vários fluxos concorrentes
  - Pode, opcionalmente, ter uma especificação de junção para modificar a sua semântica

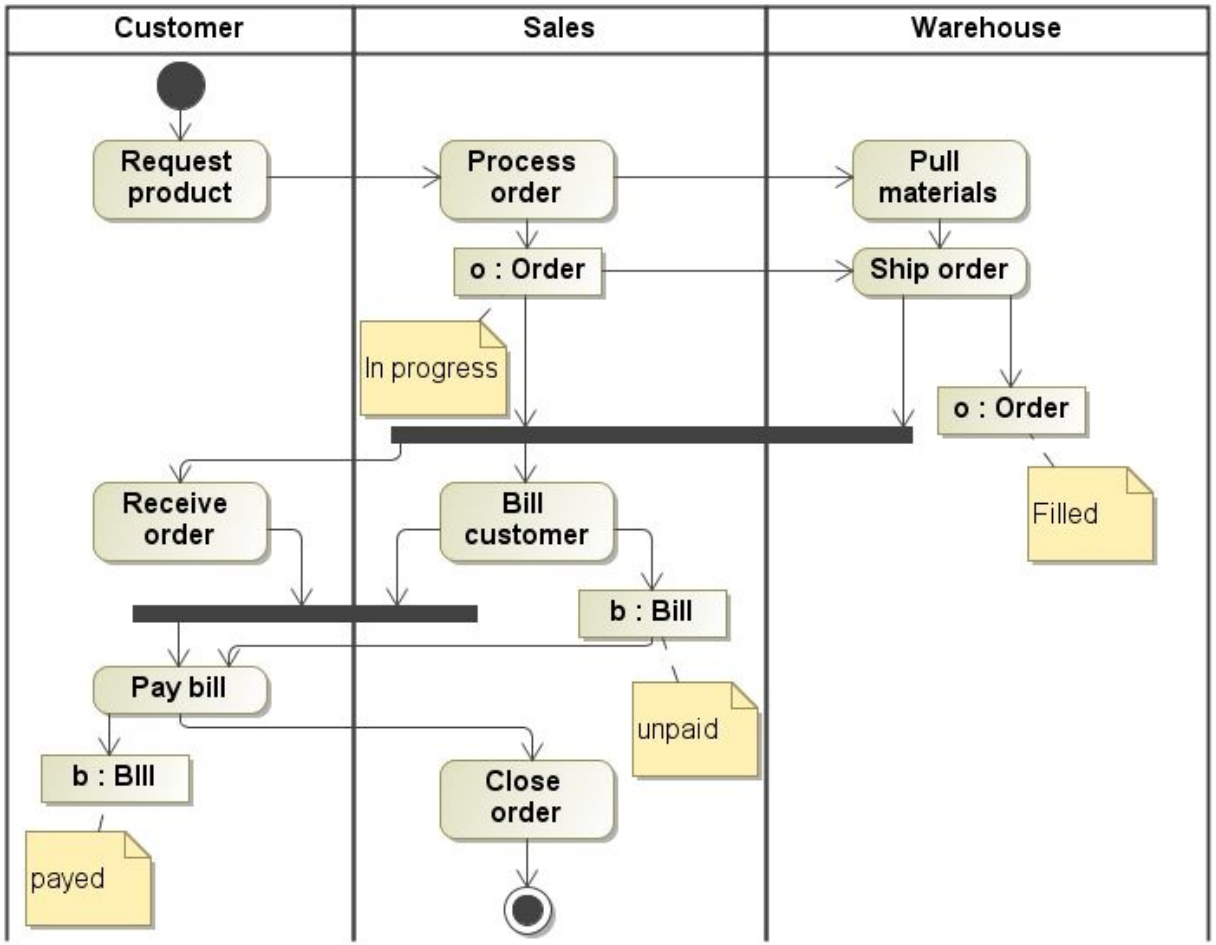
Organização de diagramas de actividade

Pistas (*Swimlanes*)

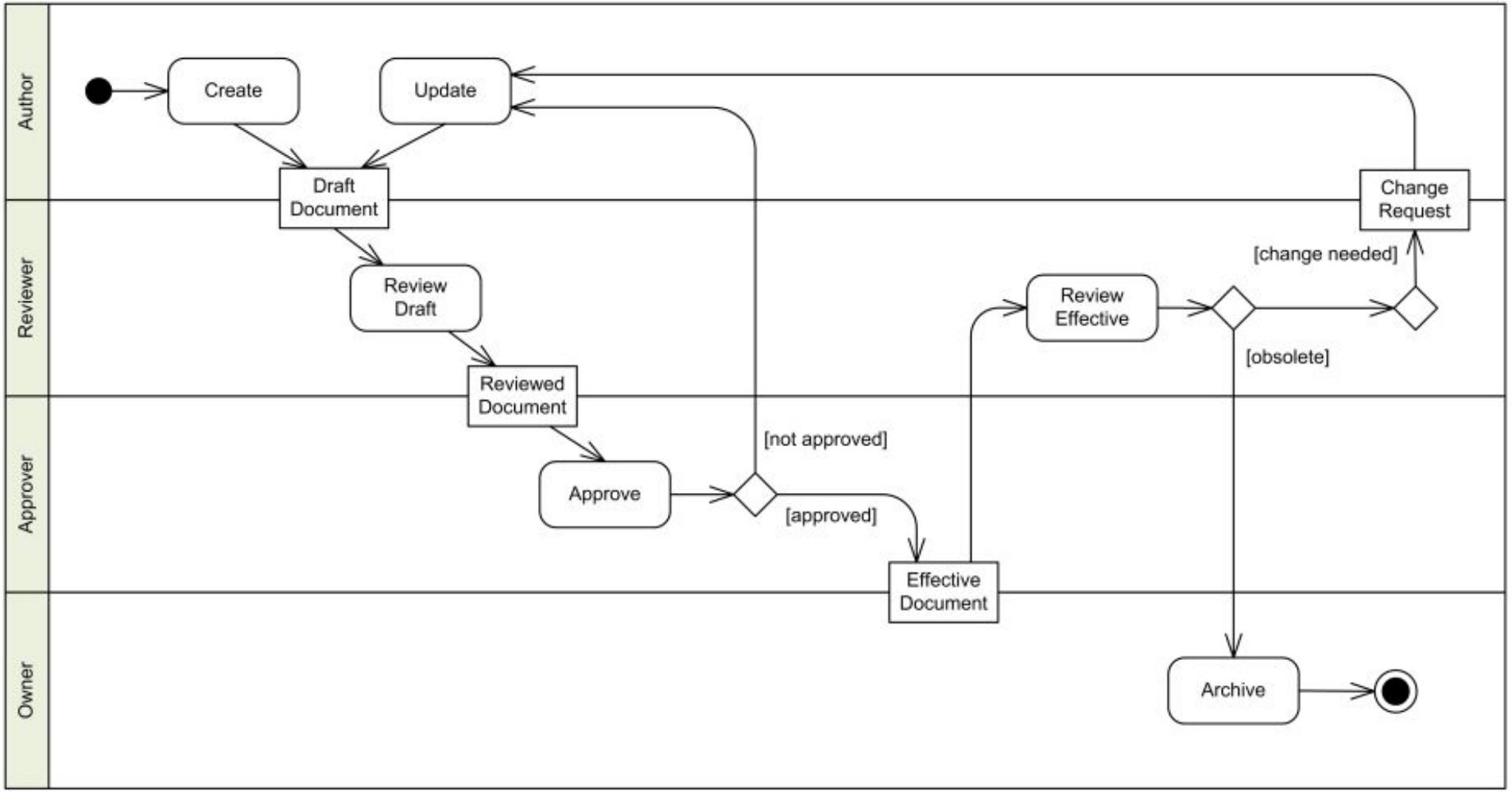
Pistas (*Swimlanes*)

- Utilizado para particionar os estados de actividade em grupos quando se modela *workflows*, *business process*, processos de software;
- Cada *swimlane* pode ser implementada por uma ou mais classes;
- Transições podem partir de uma *swimlane* para outra mas uma actividade pertence a uma só *swimlane*.

Pistas (*Swimlanes*)

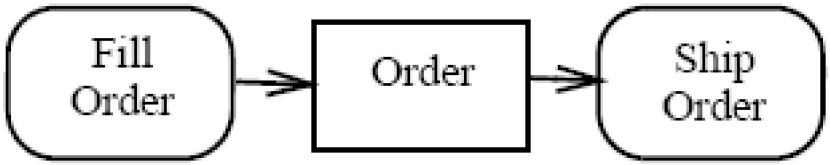


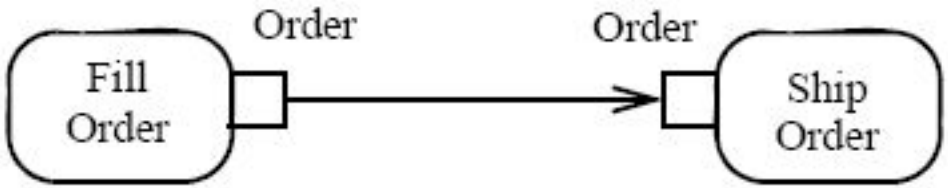
Pistas (*Swimlanes*)



Troca de informação entre actividades

Fluxo de objecto (*Object flow*)

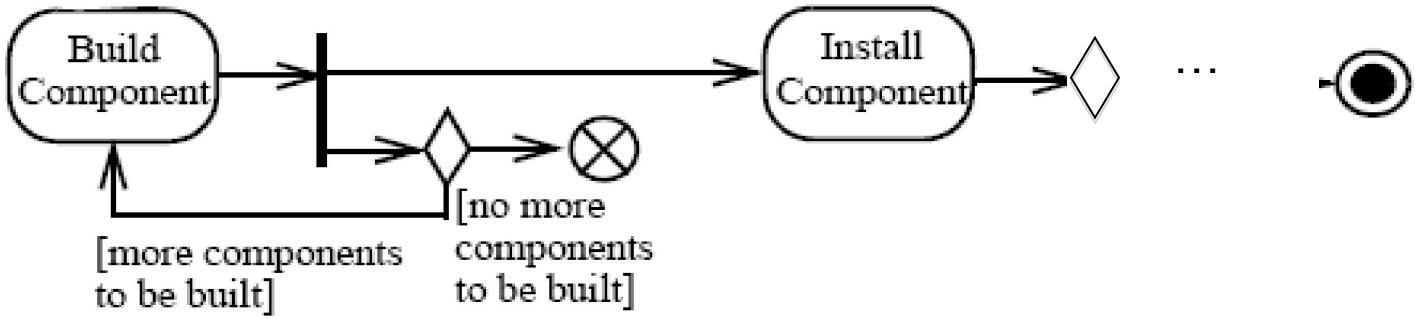




- Objectos podem estar envolvidos no fluxo de controlo associado com um diagrama de actividade
  - Os fluxos de objecto são dependências que **criam**, **removem** ou **alteram** um objecto;
  - Pode-se mostrar o estado ou os valores dos atributos do objecto.

Fluxo final + actividade final (*Final flow + final activity*)

- ***Final flow***: é um nó de controlo que termina um fluxo. Não tem qualquer efeito noutros fluxos no diagrama
- ***Final activity***: é um nó de controlo que pára todos os fluxos no diagrama. (Pode haver mais que um num diagrama.)



Sinais

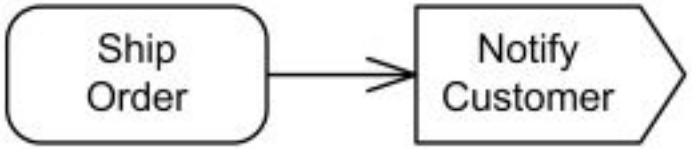
Envio de sinais

Aceitação de eventos

Aceitação de eventos temporais

Envio de Sinais

- **send signal**: acção de invocação que cria um sinal e o envia ao objecto destino, onde pode originar a execução de uma actividade ou a transição de um estado



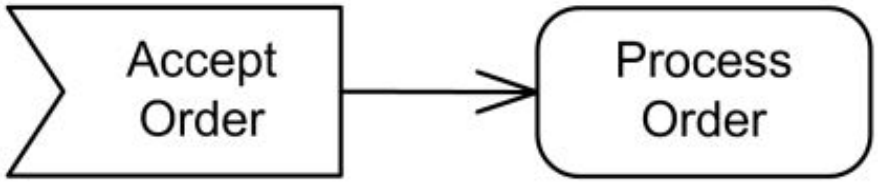
Depois de **Ship Order** cria-se o sinal **Notify Customer**

Envio de sinais

- O sinal é enviado de forma assíncrona
  - Quem envia **não fica** à espera de uma resposta, ou confirmação da recepção do sinal
- O sinal pode aceitar parâmetros de entrada, na sua criação

Aceitação de eventos

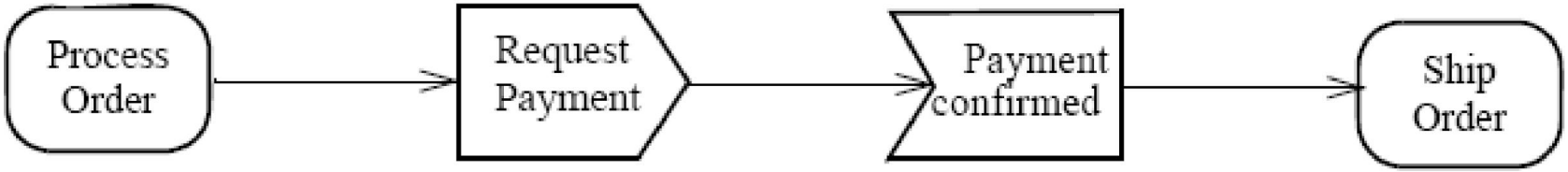
- ***accept signal***: é uma acção que espera que o evento especificado ocorra (normalmente assíncrono)



A aceitação do evento **Accept Order** causa a invocação da acção **Process Order**

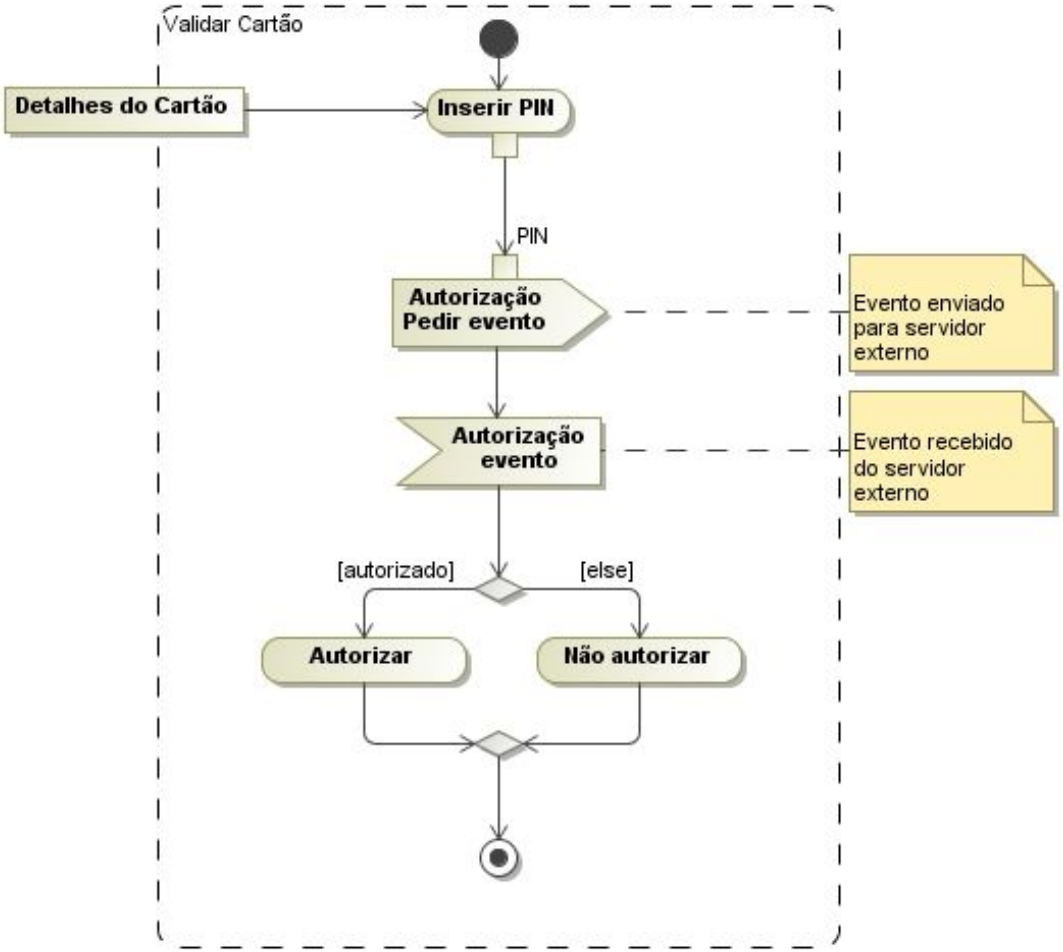
Envio de sinal / aceitação de evento

send/accept signal

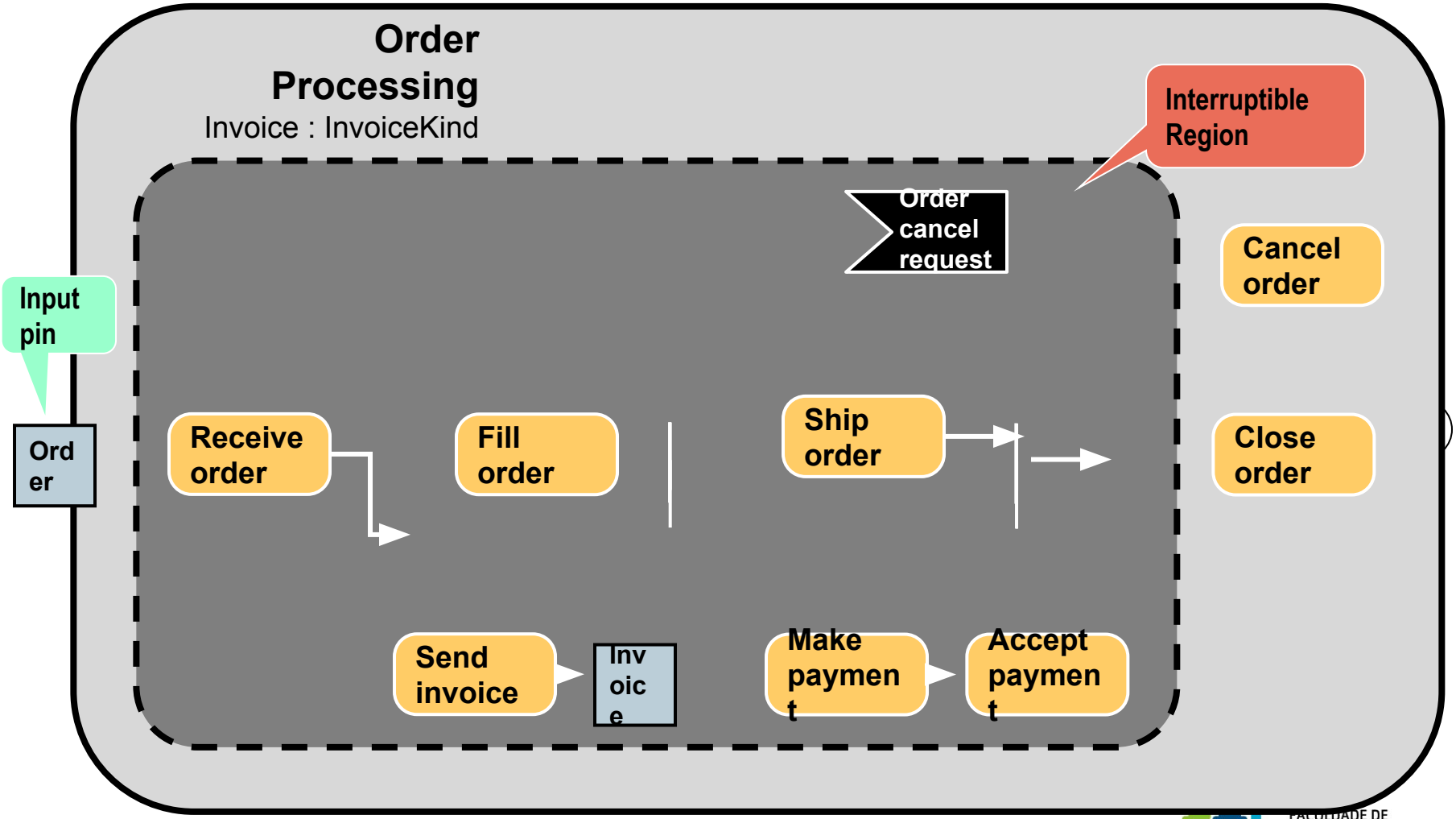


- A actividade Process Order **origina** o envio do sinal Request Payment.
- A actividade Ship Order **espera para receber** o sinal Payment Confirmed

Exemplo: Validação de cartões

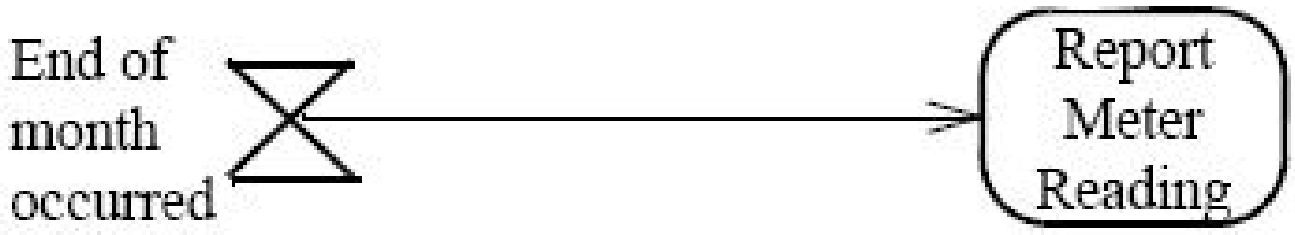


Exemplo, com regiões



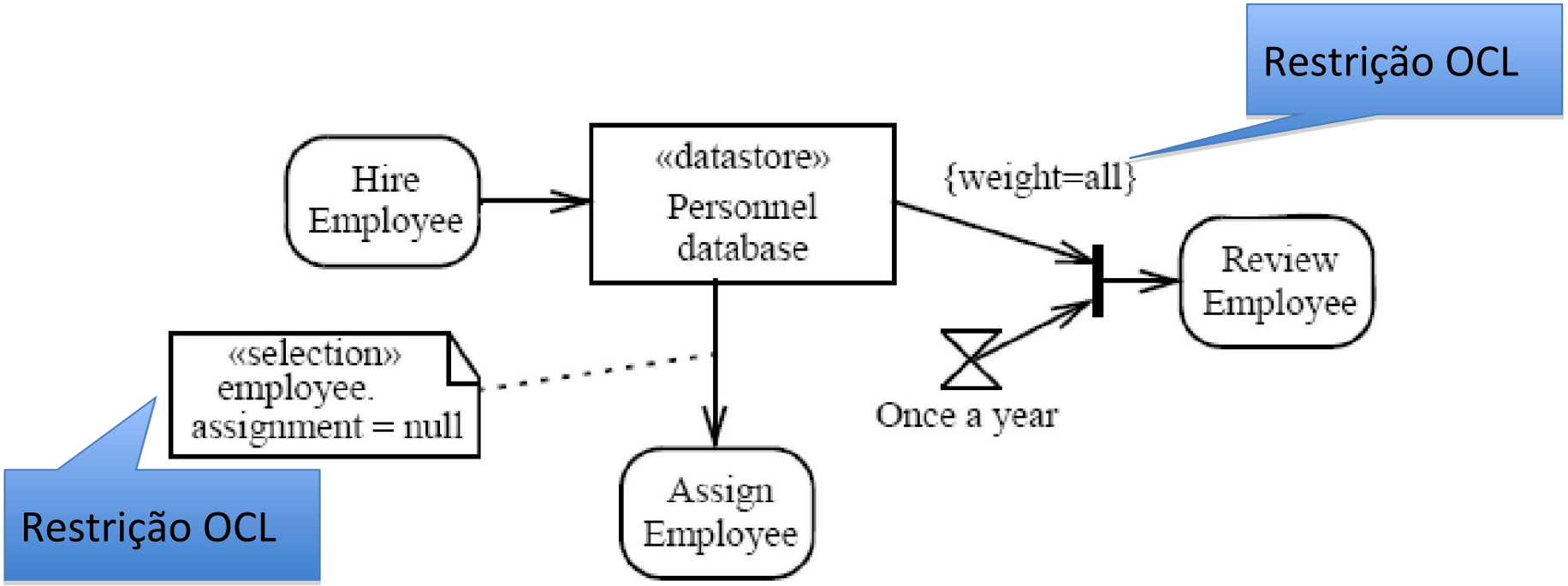
Eventos temporais (repetitivos)

- ***wait time action***



Ao fim do mês (**End of month occurred**) activa-se a actividade **Report Meter Reading**

Exemplo



Anualmente (**Once a year**) activa-se a actividade **Review Employee**, para os empregados contratados

Estilo de utilização

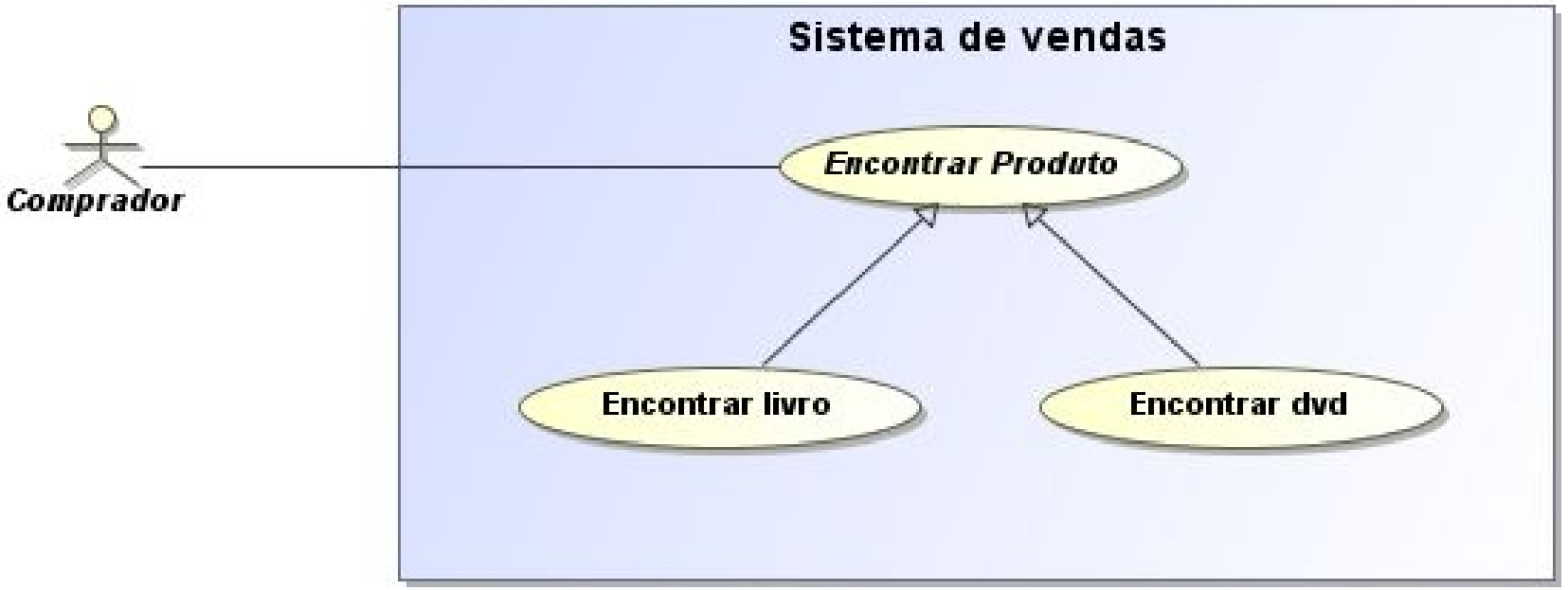
- Um bom diagrama de actividades
  - Comunica um aspecto da dinâmica do sistema em particular
  - Mostra apenas os elementos que são essenciais para compreender esse aspecto
  - Oferece apenas o nível de detalhe essencial para ser compreendido
  - Não é excessivamente minimalista, para evitar "desinformar" o leitor sobre a semântica da actividade

Algumas sugestões na criação dos diagramas de actividade

- Dê ao diagrama um nome que comunique bem o seu propósito
- Comece por modelar o propósito principal do diagrama
- Disponha os elementos de forma a minimizar as linhas cruzadas, sempre que possível
- Garanta a consistência com os restantes diagramas e outros elementos do seu modelo

Casos de uso e diagramas de actividade

Recorde o exemplo do sistema de vendas



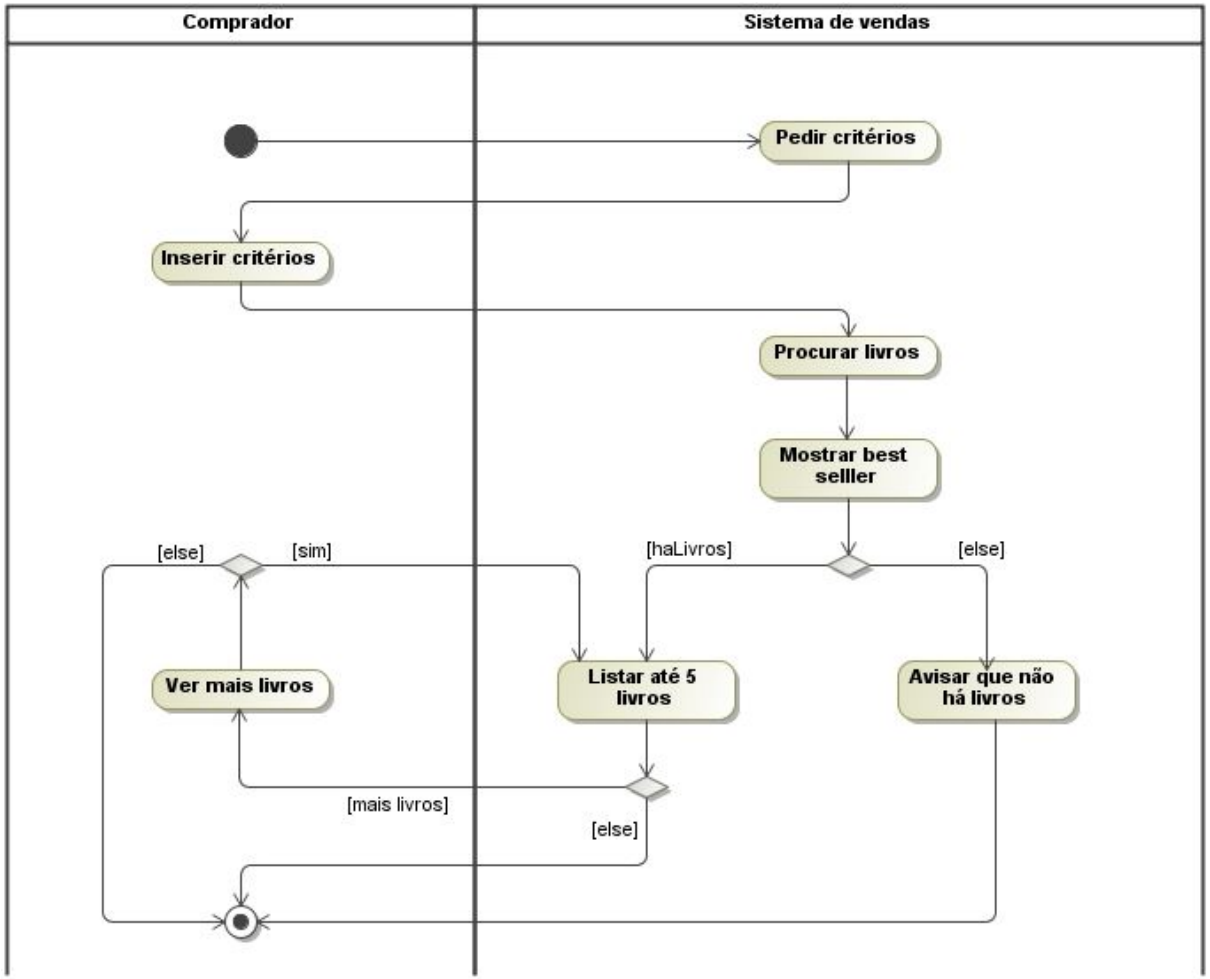
Caso de uso Encontrar Produto

Caso de Uso: *Encontrar produto*
Descrição: O comprador procura e visualiza o estado de uma encomenda no sistema
Actor principal: *Comprador*
Actores secundários: Nenhum
Pre-condições: Nenhuma
Fluxo principal: 1. O caso de uso começa quando o comprador selecciona a opção encontrar produto. 2. O sistema pede ao comprador para indicar o critério de pesquisa a usar. 3. O comprador introduz os critérios de pesquisa. 4. O sistema procura produtos que satisfazem os critérios de pesquisa do comprador. 5. Se o sistema encontra produtos que satisfazem os critérios 5.1. O sistema apreenta uma lista de produtos que satisfazem os critérios. 6. Caso contrário 6.1. O sistema indica ao comprador que nenhum produto foi encontrado.
Pos-condições: Nenhuma
Fluxos alternativos: Nenhum

Caso de uso encontrar livro

Caso de Uso: Encontrar Livro
Descrição: O comprador procura e visualiza o estado de uma encomenda no sistema
Actor principal: *Comprador*
Actores secundários: Nenhum
Pre-condições: Nenhuma
Fluxo principal: 1. (o1.) O caso de uso começa quando o comprador selecciona a opção encontrar livro. 2. (o2.) O sistema pede ao comprador para indicar o critério de pesquisa do livro a usar, que deve incluir o autor, título, ISBN, ou tópico. 3. O comprador introduz os critérios de pesquisa. 4. (o4.) O sistema procura livros que satisfazem os critérios de pesquisa do comprador. 5. (o5.) Se o sistema encontra livros que satisfazem os critérios 5.1. O sistema mostra o best seller. 5.2. (o5.1.) O sistema apreenta uma lista com até 5 livros que satisfazem os critérios. 5.3. Para cada livro o sistema mostra o título, autor, preço e ISBN 5.4. Enquanto houver mais livros para mostrar, o sistema dá ao comprador a possibilidade de pedir a próxima página com mais livros. 6. (6.) Caso contrário 6.1. O sistema mostra o best seller. 6.2. (6.1.) O sistema indica ao comprador que nenhum produto foi encontrado.
Pos-condições: Nenhuma
Fluxos alternativos: Nenhum

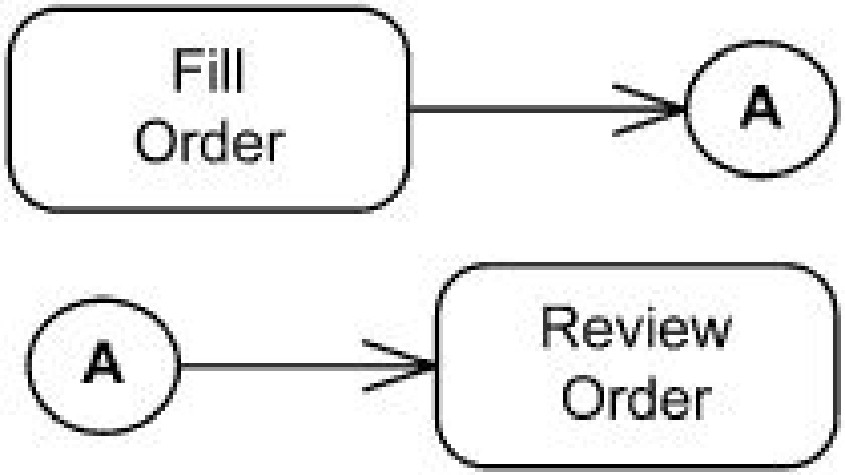
Diagrama de actividades Encontrar Livro



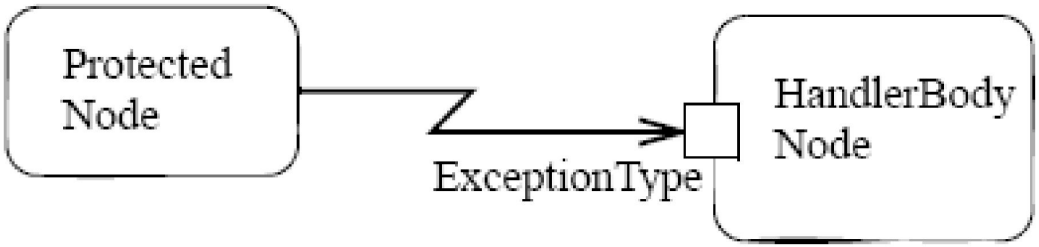
Conceitos avançados em diagramas de actividade

Conector

- Usado para evitar uma transição comprida
- Não afecta o modelo



Conceitos avançados: *exception handler*

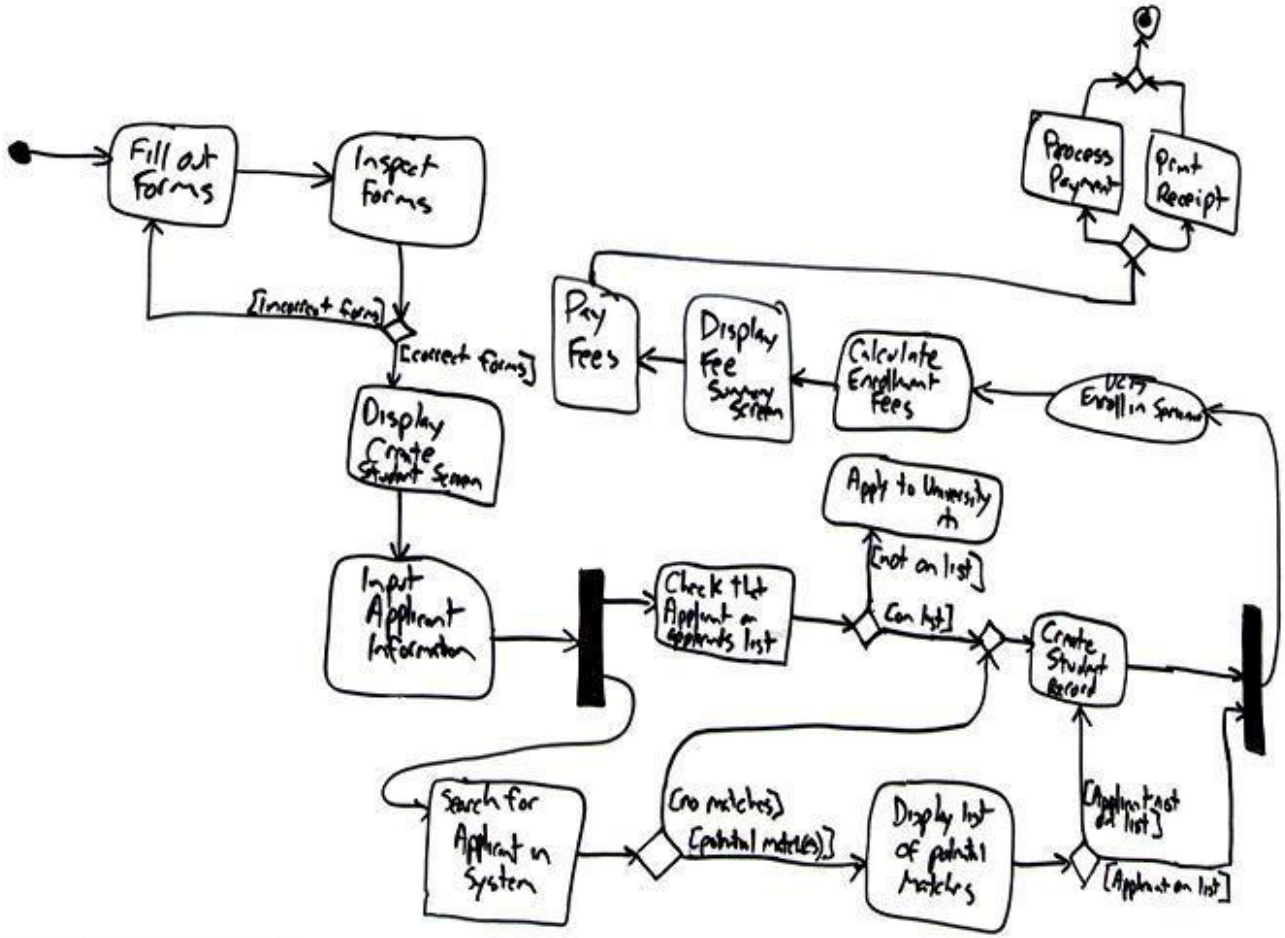


- *If an exception occurs during the execution of an action, the execution of the action is abandoned and no regular output is generated by this action. If the action has an exception handler, it receives the exception object as a token. If the action has no exception handler, the exception propagates to the enclosing node and so on until it is caught by one of them. If an exception propagates out of a nested node (action, structured activity node, or activity), all tokens in the nested node are terminated. The data describing an exception is represented as an object of any class.*

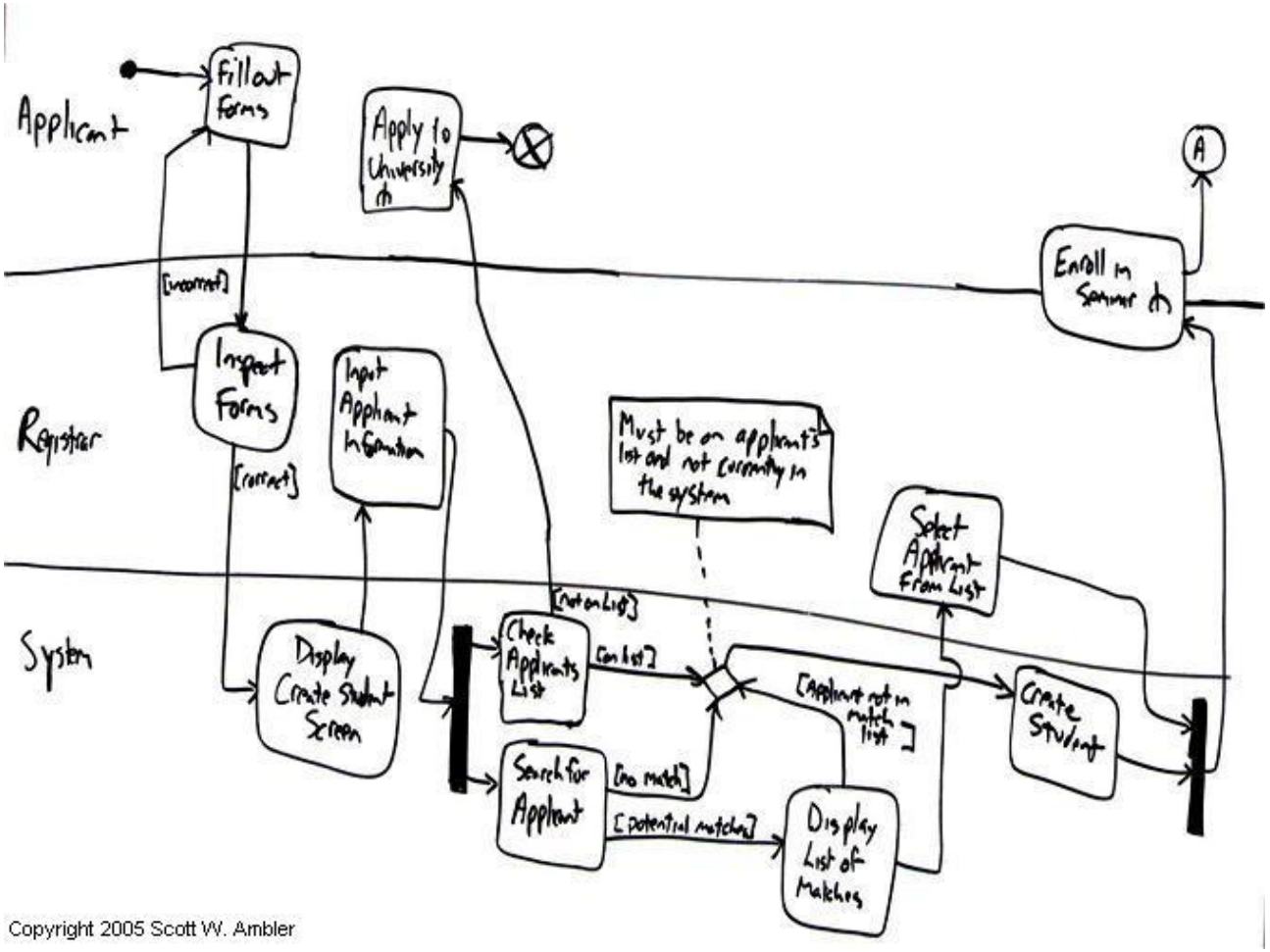
[http://www.uml-diagrams.org/activity-diagrams.html#partition]

Exercícios

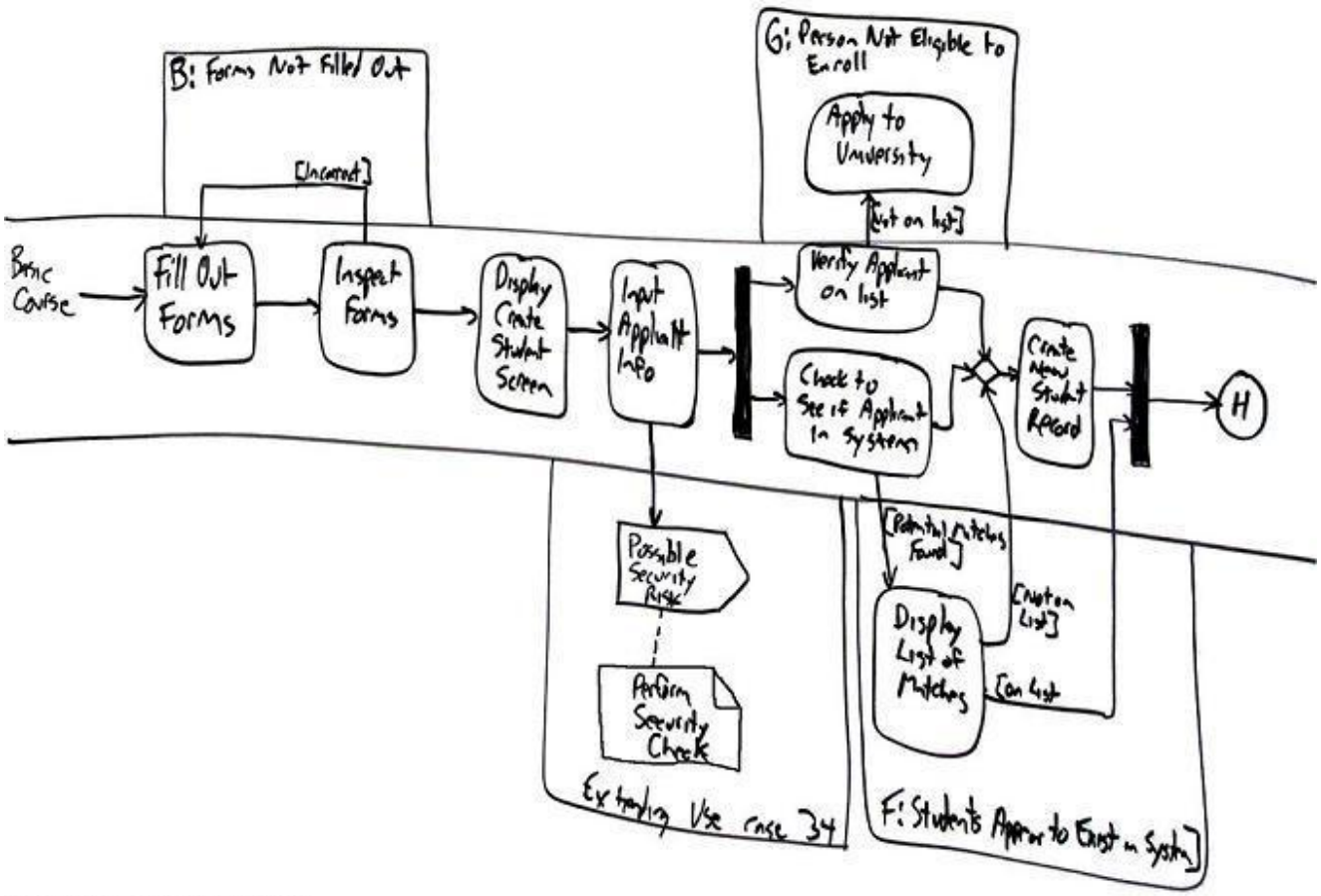
Este diagrama está correcto?

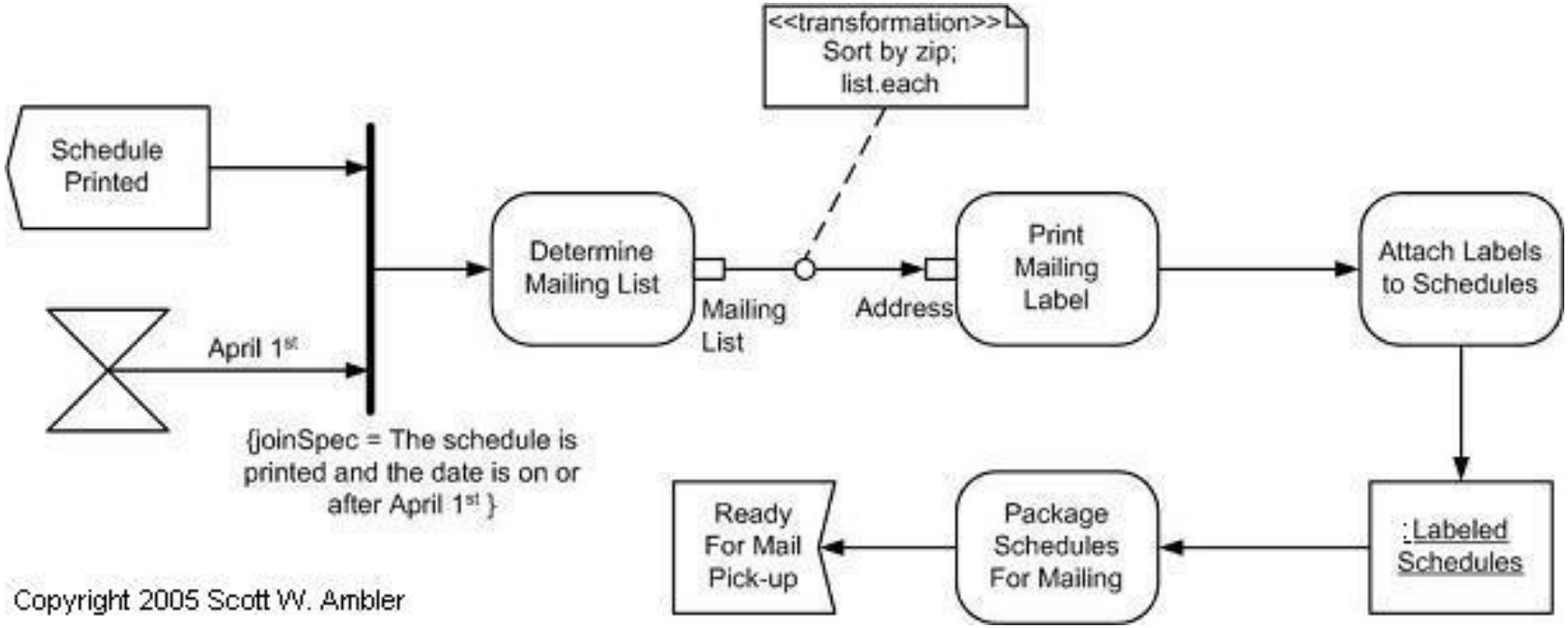


E este?


Applicant
Register
System
Fill out forms
Apply to University
[incorrect]
Inspect Forms
[correct]
Input Applicant Information
Display Create Student Screen
Must be on applicant's list and not currently in the system
[not on List]
Check Applicants List
[on list]
Search for Applicant
[no match]
[Potential matches]
Display List of Matches
Select Applicant From List
[Applicant not in match list]
Create Student
Enroll in Seminar
A
Copyright 2005 Scott W. Ambler

E este?




<<transformation>>
Sort by zip;
list.each
Schedule Printed
April 1st
{joinSpec = The schedule is printed and the date is on or after April 1st }
Determine Mailing List
Mailing List
Address
Print Mailing Label
Attach Labels to Schedules
:Labeled Schedules
Package Schedules For Mailing
Ready For Mail Pick-up
Copyright 2005 Scott W. Ambler

E quando definimos diagramas de actividade separados para cenários alternativos?

- *Extension points* no diagrama de casos de uso
- *Pattern Specifications* para diagramas de actividade?

Pattern Specifications (1)

- Permitem a formalização da reutilização de modelos
- Baseado em UML
- Um *PS* descreve um padrão de estrutura ou comportamento
  - Definido sobre os papéis (*roles*) desempenhados pelos participantes nesse papel
  - *Roles* são precedidos por uma barra vertical "|"

Pattern Specifications (2)

- Um PS pode ser instanciado atribuindo elementos de modelo concretos para desempenhar os papéis (*roles*) identificados
- Um *role* é uma especialização de uma metaclasse UML restringida por propriedades adicionais que qualquer elemento a desempenhar o role deve possuir
  - Metaclasse: uma classe cujas instâncias são classes