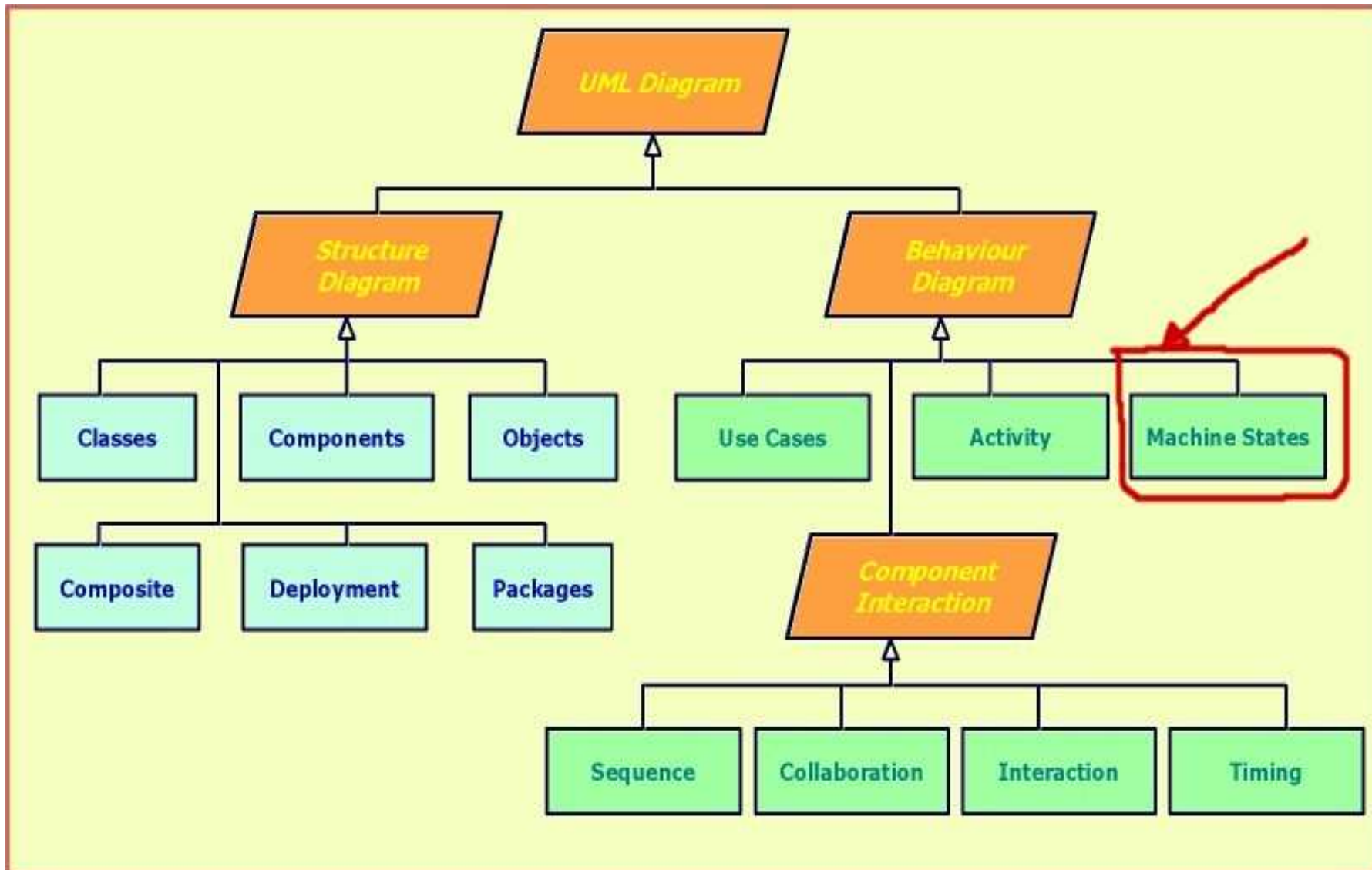
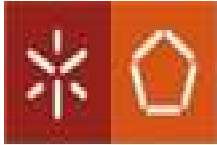




Diagrama representativo de uma Máquina de Estados (DME)





DIAGRAMAS DE MÁQUINAS DE (TRANSIÇÃO DE) ESTADOS

▣ Para que se compreenda a verdadeira importância dos DME em UML, é, em rigor, necessário compreender a sua **génese** e **objectivos**, em especial apresentando as **teorias/modelos** nos quais se baseiam.

▣ **Génese:** São adaptações UML de notações há muito existentes para a descrição do comportamento de sistemas sob a forma de Autómatos (máquinas que funcionam sozinhas em resposta a entradas de um dado tipo), que em cada momento se encontram num estado interno que representa a sua memória do passado e o seu conhecimento do presente para que possa transitar correctamente para outro estado quando surge a **entrada** ie. **evento seguinte**;

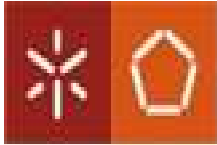
▣ As adaptações visam permitir que UML possa especificar o comportamento de dispositivos físicos (**máquinas hardware**) e de entidades lógicas ou “**máquinas software**” (numa perspectiva OO, **certos objectos**);

▣ **Vamos definir alguns conceitos fundamentais sem os quais os DMEs não podem ser compreendidos nem correctamente usados.**



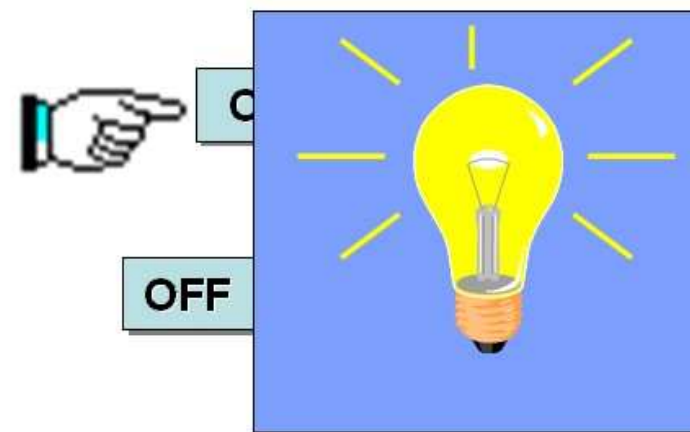
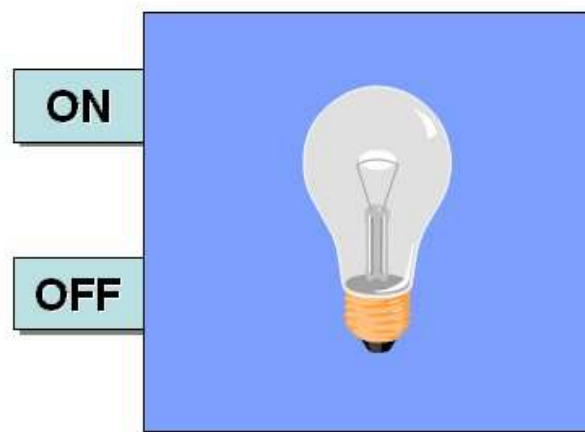
AUTÓMATOS (Máquinas de Estados Finitos)

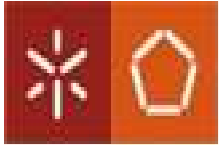
- ▶ São máquinas cujo comportamento é uma consequência não apenas da última entrada mas também de todo o passado de entradas (visto como a sequência de entradas realizadas, por exemplo máquina de café);
- ▶ Caracterizam-se por, a cada momento, se encontrarem num estado interno que “representa” toda a sua experiência passada (ou seja, o resultado da sequência de todas as entradas registadas). Comportamento consiste em transitar de estado em estado (estados são em número finito).
- ▶ Interactuamos com estes autómatos diariamente, porque diariamente usamos as máquinas de venda de qualquer coisa, cf. as máquinas de venda de bebidas, de chocolates, etc.; usamos também as ATM, usamos as bombas de gasolina e as estações de lavagem automática de carros.
- ▶ Há, porém, muitas “máquinas software” (classes) que produzem “objectos” que possuem características de comportamento, passivo ou activo, muito semelhantes a estas máquinas baseadas em electrónica e mecânica.
- ▶ Vejamos alguns conceitos importantes sobre autómatos.



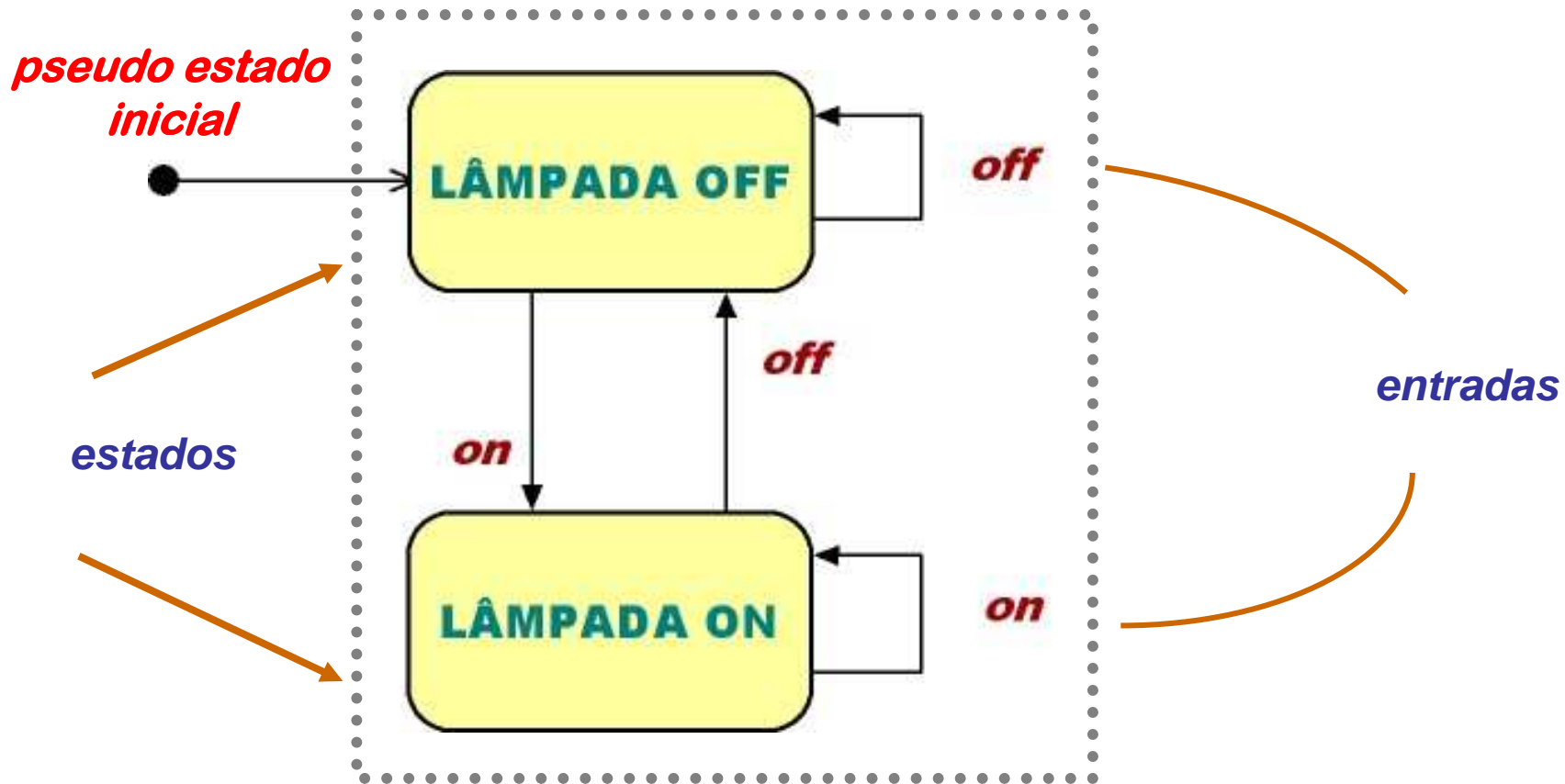
AUTÓMATOS

- ▶ São máquinas cujo **comportamento** é uma consequência não apenas da **última entrada** mas também de **todo o passado** (visto como a sequência de entradas realizadas).
- ▶ Caracterizam-se por possuírem um **estado interno** que “representa” toda a sua **experiência passada** (**comportamento** => **transição de estado**).

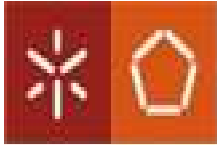




▣ Representação gráfica do **comportamento do autómato** em termos de **estados, entradas e transições entre estados**.



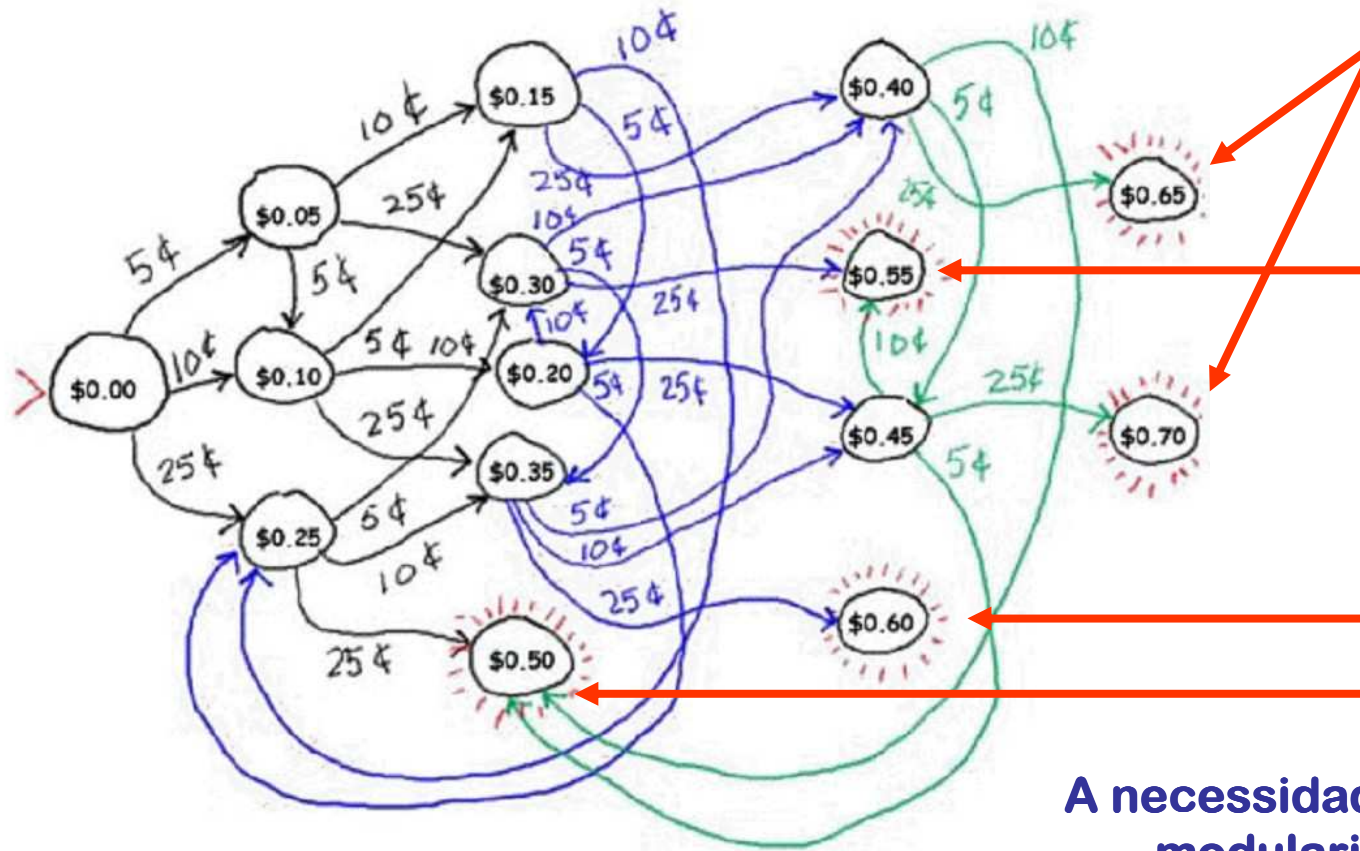
start + on + on + off + on = ? (estado actual depende do passado)



- ▶ Torna-se pois interessante e importante colocar a questão: Seríamos neste momento capazes de conceber, usando uma notação rigorosa (nunca podendo realizar tal implementação), uma máquina de venda de chocolates ao preço fixo de 50 cêntimos ? Como descreveríamos o que se passa no interior de tal máquina, à medida que vão sendo introduzidas moedas, de tal forma que a máquina aceite moedas até ao momento em que a sequência de valores das moedas introduzidas conduza a máquina a um estado interno em que a mesma passe a aceitar a ordem de fornecer o produto e, posteriormente, eventualmente dar um troco ?
- ▶ Claro que se tivéssemos introduzido 5c depois de termos introduzido 25c, a máquina terá que “saber” que o total inserido é de 30c, e mais do que isso, que não dá para comprar o chocolate, ou seja, o seu estado actual não é “aceitável” para a conclusão com sucesso da tarefa. Mas o que é importante compreender neste contexto, é que, se foram introduzidos 10c e depois 25c, então a máquina terá que “saber” que “actualmente capitaliza” 35c, e que, ainda assim, não se encontra num estado “aceitável” para a efectivação da compra.



▣ Representação gráfica dos estados possíveis, entradas e transições de estado de uma **máquina de venda de chocolates a 50 c**, até se atingir uma quantia que permita **comprar um chocolate**.

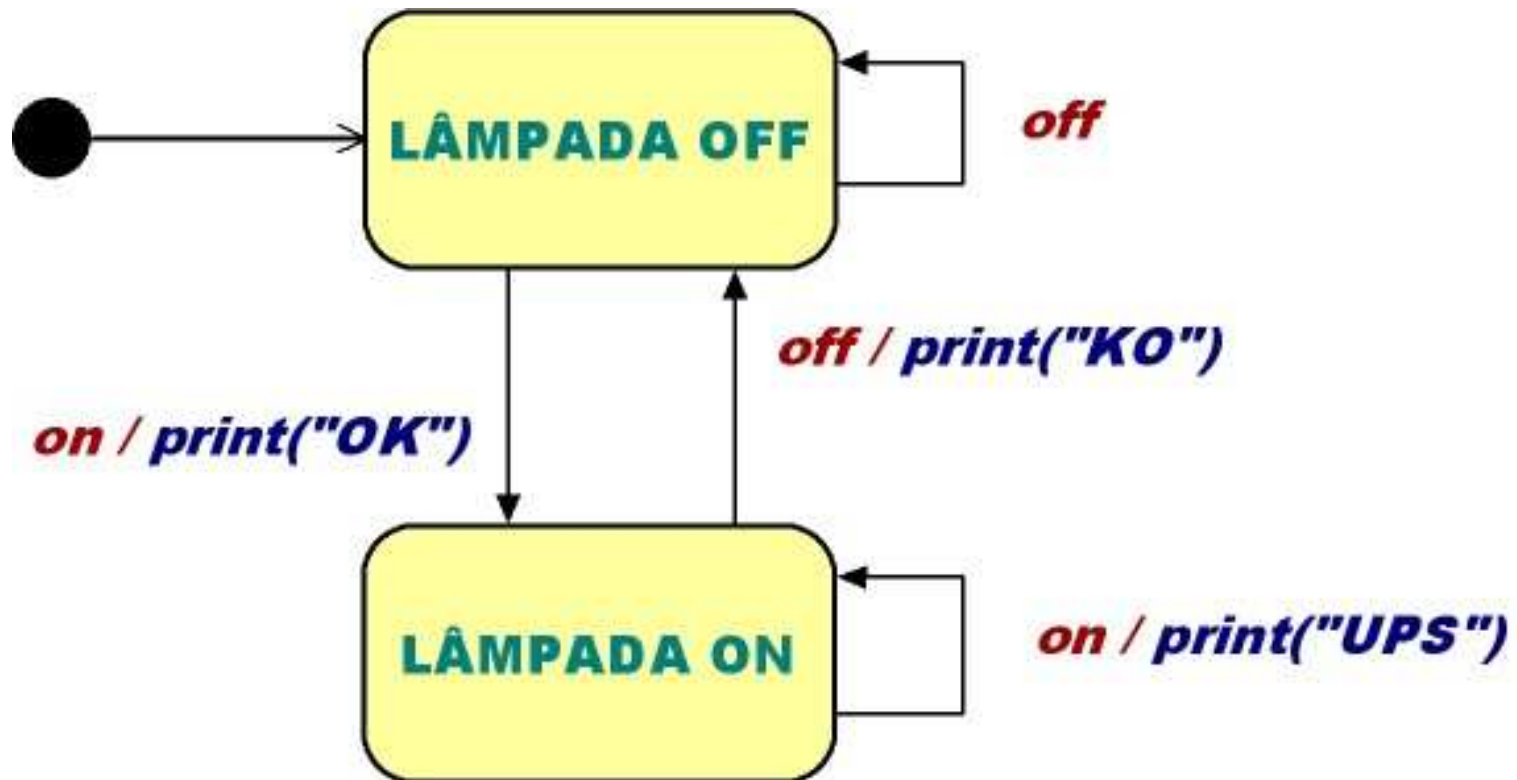


Estados aceitáveis para continuação do comportamento da máquina
(exº *comprar*)

A necessidade de estrutura e modularidade é óbvia

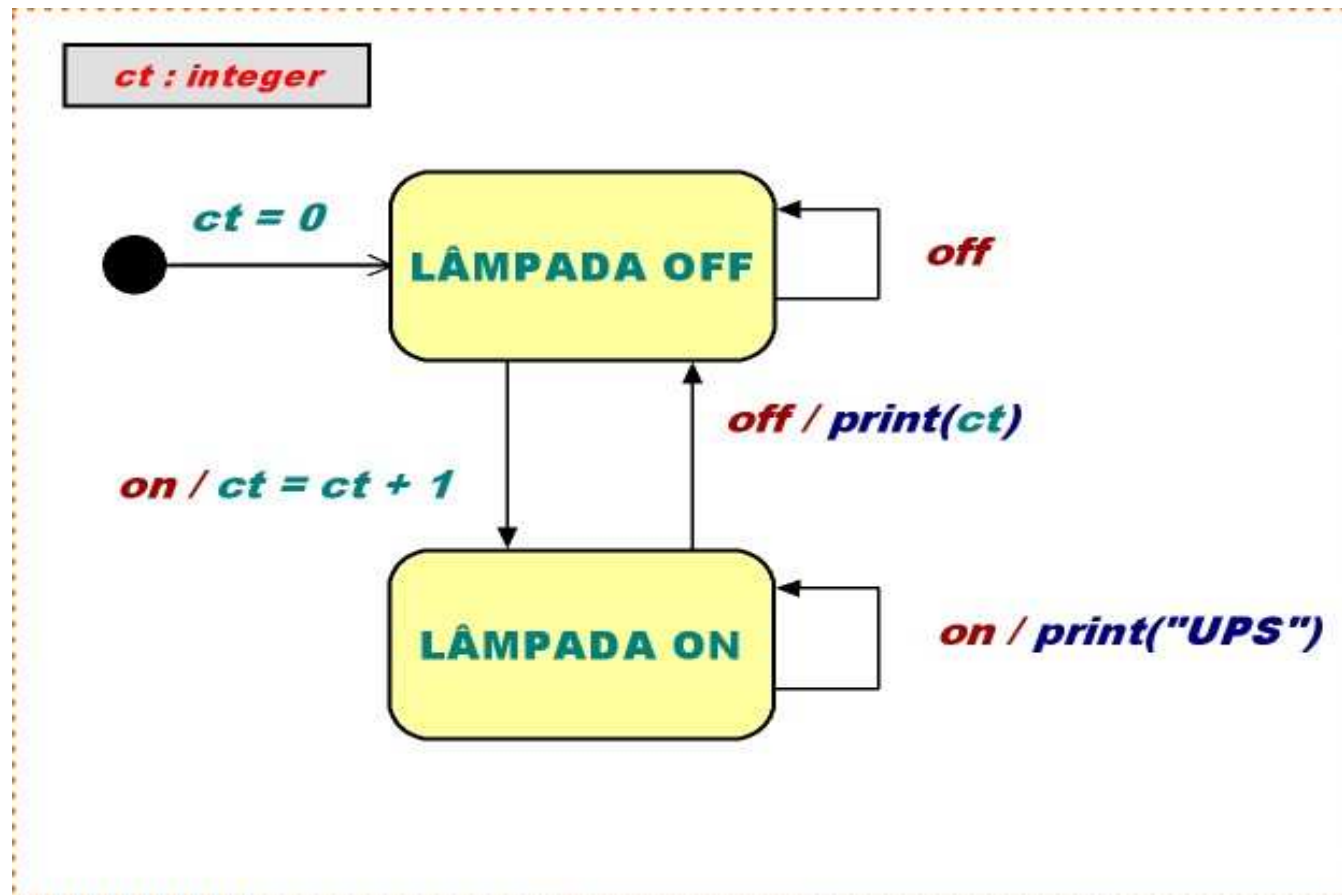


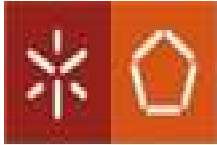
▣ Quando muda de estado, um autómato pode gerar “**outputs**” (resultados de acções) que se descrevem associados às **entradas-eventos**, tal como em **entrada / acção**





▣ As acções associadas às transições podem ser quaisquer. Se associarmos ao autómato **variáveis** (“**extended state automaton**”), as acções podem ter a ver com o controlo dos valores dessas variáveis.





☐ Uma máquina de estados estendida (**Mealy**) define-se como:

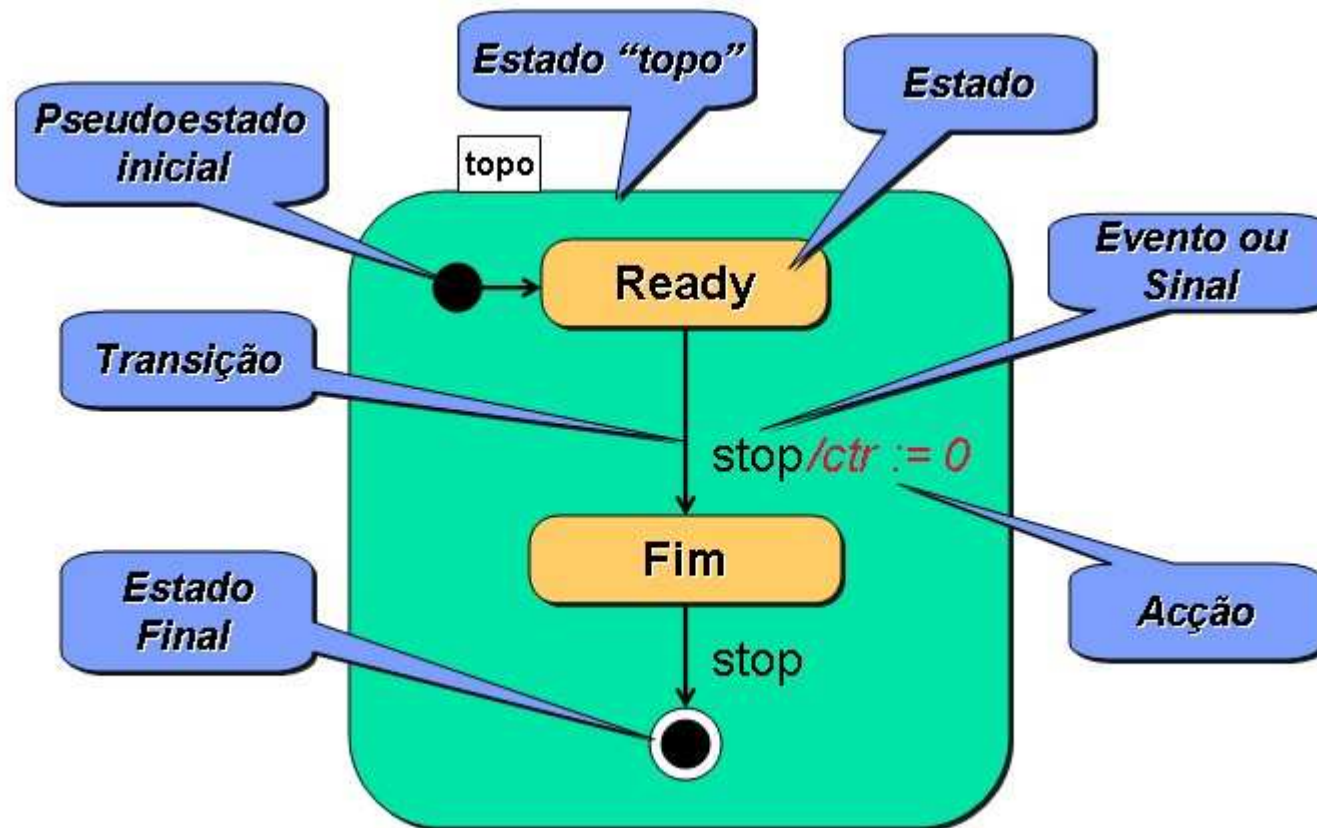
- ▶ um conjunto de sinais de entrada (alfabeto de input)
- ▶ um conjunto de sinais de saída (alfabeto de output)
- ▶ um conjunto **finito** de estados (cf. FSM – **finite state machine**)
- ▶ um conjunto de transições (eventou ou sinal e acção)
- ▶ um conjunto de variáveis
- ▶ um estado inicial
- ▶ um conjunto de estados finais (se o autómato termina)

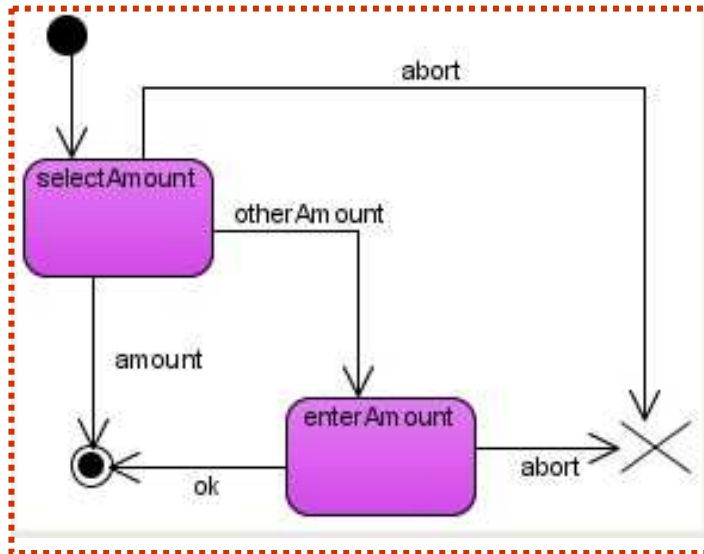
☐ Máquinas de Estados apenas podem modelar **comportamento discreto**, ou seja, **não contínuo e baseado na reacção a eventos**, **comportamento reactivo discreto** (que não está sempre a ocorrer, ao contrário de um **motor de carro** ou **sistema de ar condicionado**).

☐ Ao contrário do estado de um objecto OO, que corresponde aos valores dos seus atributos num dado momento, os estados de um autómato são mais abstractos porque **diferentes estados implicam diferentes reacções a eventos** (**comportamento diferente**).



▣ Diagrama de Estados básico de UML





Transições representam passagens de um estado a outro pela ocorrência de um **evento**.

Associado ao **evento** de uma transição podemos ter **parâmetros** e **acções**.

Há transições que são **condicionadas**, ou seja, **guardadas** por uma condição entre [...]. Se **true** há transição senão ...

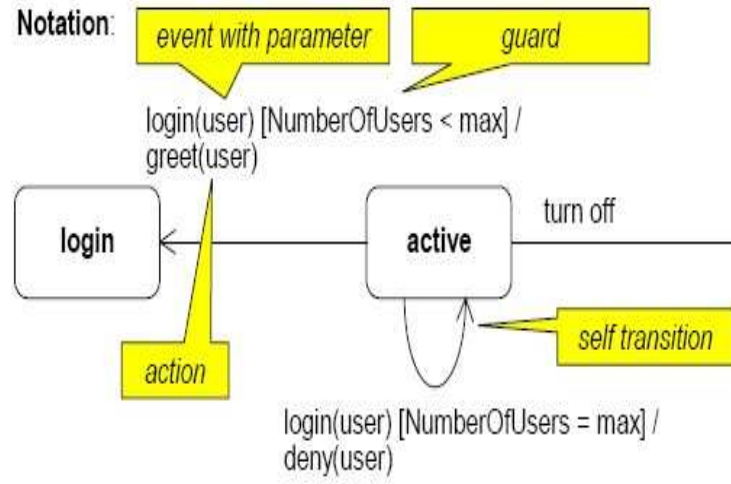
Quando uma máquina está num dado **estado** e ocorre um **evento**, **apenas uma transição de saída pode ser tomada**.

Eventos, transições e actividades são realizados de forma instantânea.

Forma geral:

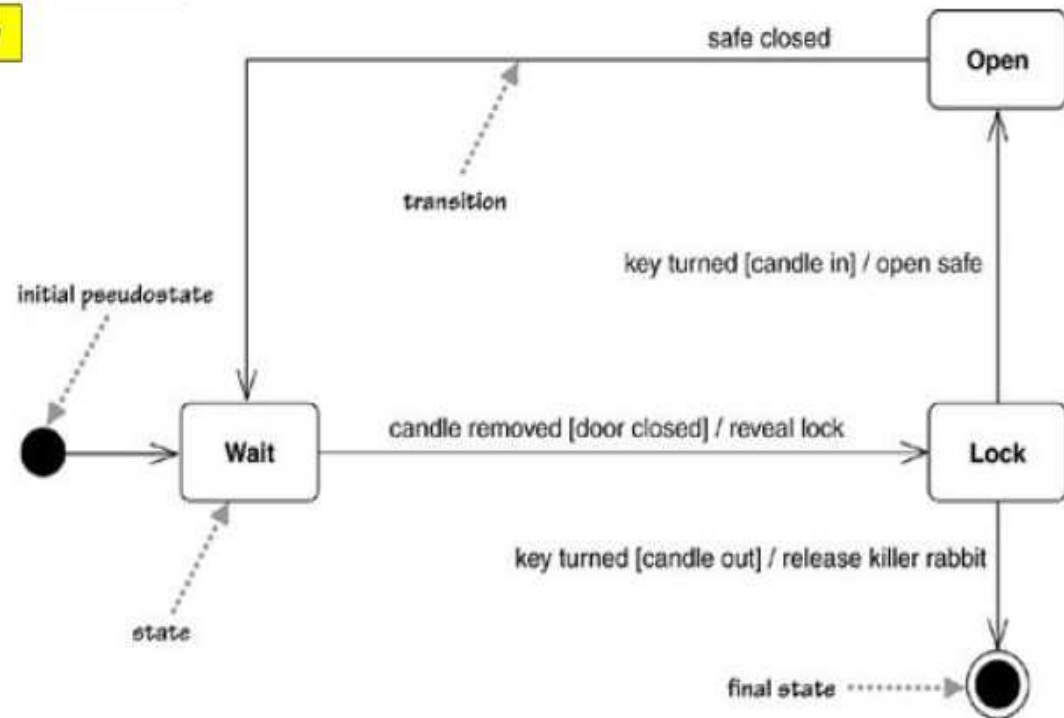
evento(pars) [guarda] / acção

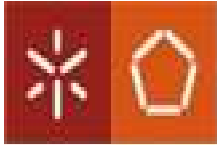
Qualquer das 3 partes é opcional !



Um simples login ...

Um cofre muito especial





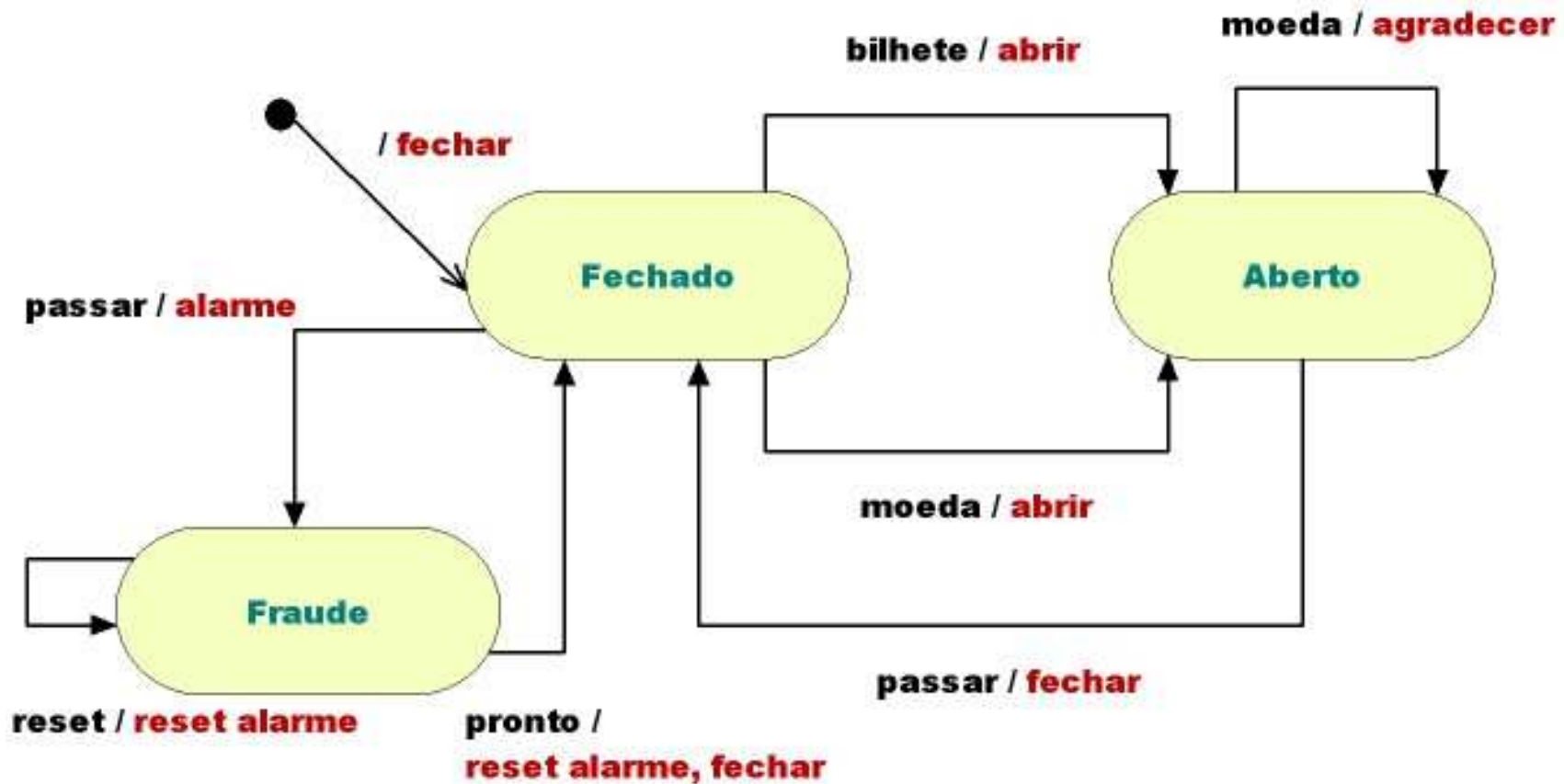
☐ Um **estado** é uma **situação particular** dentro das várias situações possíveis no ciclo de vida de um autómato, durante a qual o autómato possui dadas propriedades (cf. **aberto, fechado, feliz, infeliz**), satisfaz alguma condição (cf. **transitável ou não, válido ou inválido**), realiza alguma actividade (cf. **faz soar o alarme, ou faz pisca-pisca**), ou então apenas espera a ocorrência de algum evento válido.

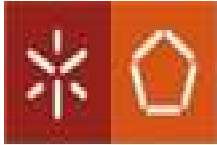
☐ Tal como foi dito antes, **os estados relevantes de um DME são aqueles que respondem de forma diferente aos eventos que podem ocorrer** (ver exemplo seguinte);

☐ **Estes autómatos são determinísticos**, pelo que, tal como se disse antes, a especificação apenas estará correcta se, partindo-se de um qualquer estado actual, apenas 1 estado seguinte seja possível de atingir dadas as combinações de eventos e guardas. Esta propriedade é muito importante e designa-se por **determinismo**.

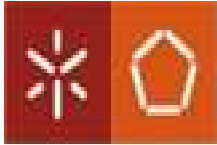


DME: Torniquete de entrada no Metro ou num Estádio





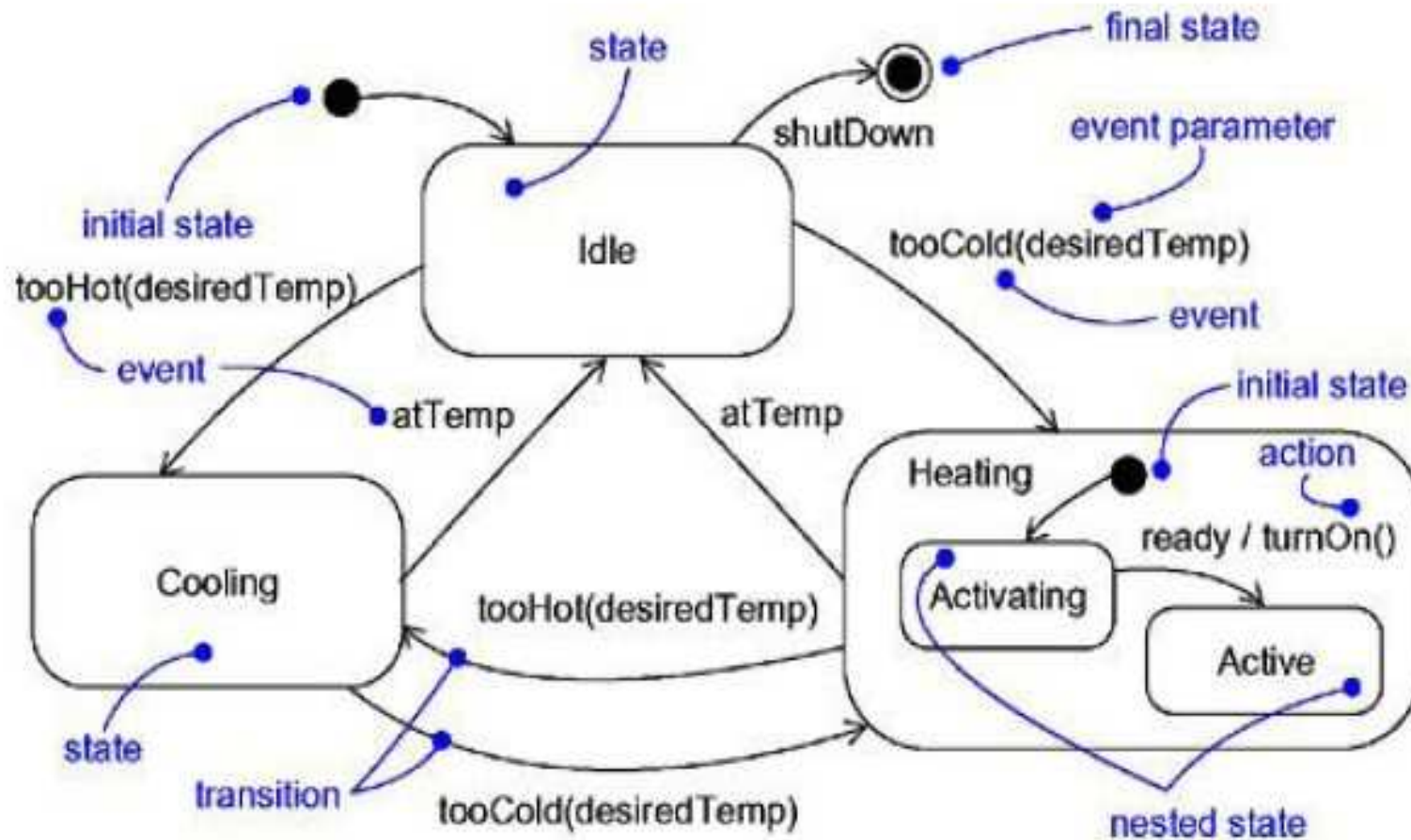
```
public static Estado estado = Estado.Fechado;
public enum Estado { Fechado, Aberto, Fraude };
public enum Evento { bilhete, moeda, passar, pronto, reset };
public void fechar();
public void abrir();
public void passar();
public void alarme();
.....
public void Transicao(Evento e) {
    switch(estado) {
        case Estado.Fechado:
            switch(e) {
                case Evento.moeda: { estado = Estado.Aberto; abrir(); break; }
                case Evento.bilhete: { estado = Estado.Aberto; abrir(); break; }
                case Evento.passar: { alarme(); break; }
            }
        break;
        case Estado.Aberto:
            switch(e) {
                case Evento.moeda: { escrever("obrigado"); break; }
                case Evento.passar: { fechar(); break; }
            }
        break; .....
    }
}
```



Implementação em tabela

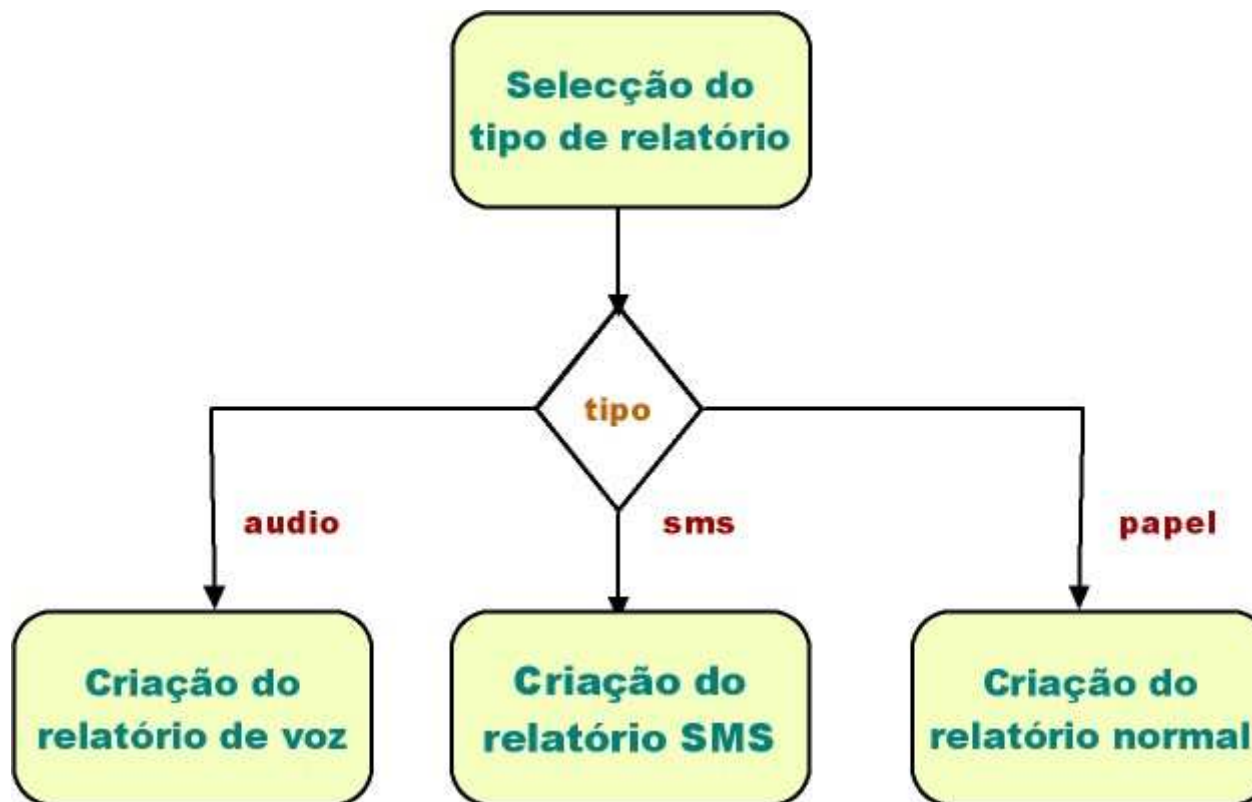
Estado Origem	Estado Destino	Evento	Guarda	Actividade
<i>pseudo</i>	<i>Fechado</i>	<i>null</i>	true	<i>fechar</i>
<i>Fechado</i>	<i>Aberto</i>	<i>bilhete</i>	true	<i>abrir</i>
<i>Fechado</i>	<i>Aberto</i>	<i>moeda</i>	true	<i>abrir</i>
<i>Fechado</i>	<i>Fraude</i>	<i>passar</i>	true	<i>alarme</i>
<i>Aberto</i>	<i>Fechado</i>	<i>passar</i>	true	<i>fechar</i>
<i>Fraude</i>	<i>Fraude</i>	<i>reset</i>	true	<i>reset alarme</i>
<i>Fraude</i>	<i>Fechado</i>	<i>pronto</i>	true	<i>reset alarme, fechar</i>

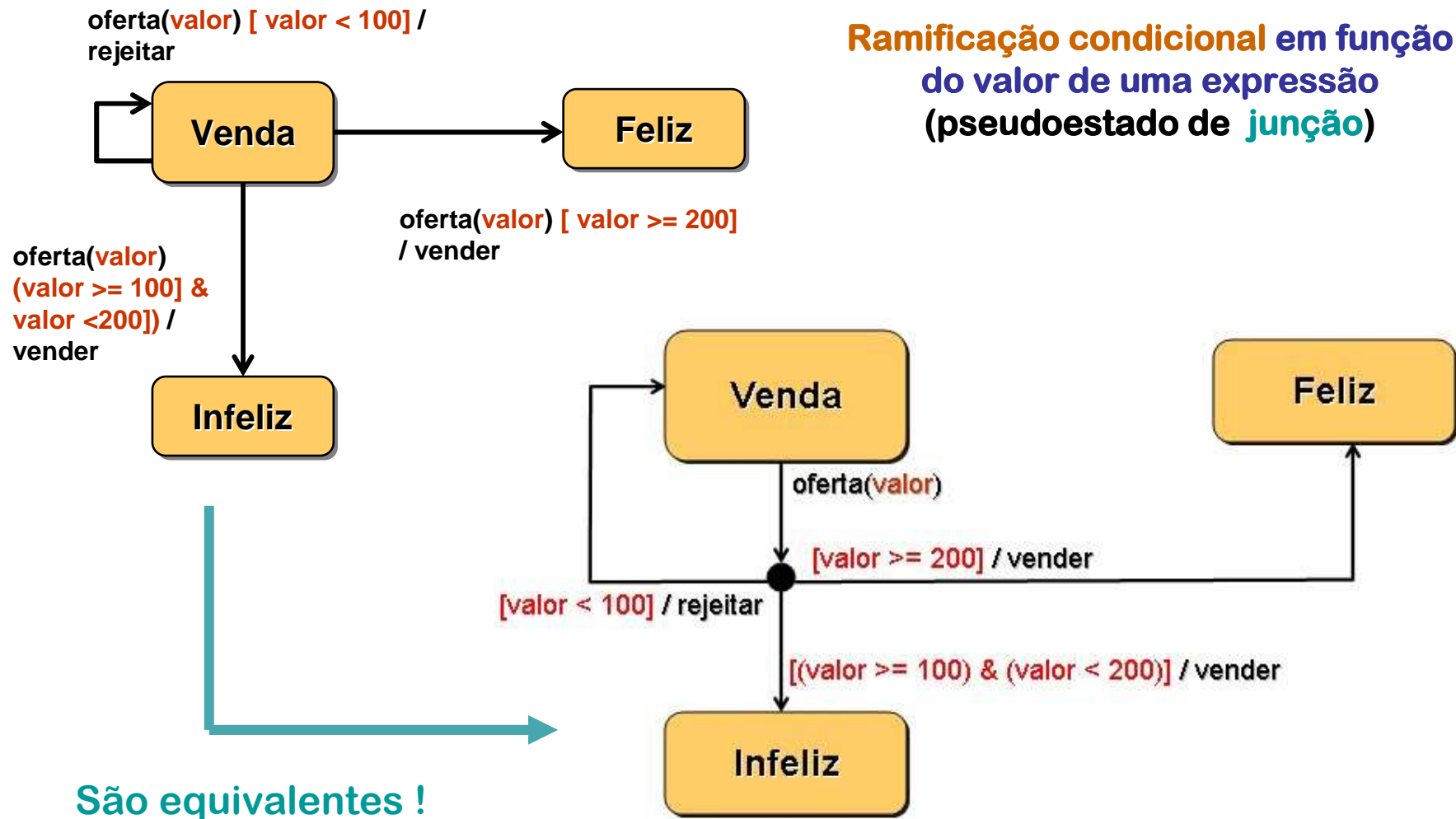
A implementação em tabelas é muito importante quando se pretende representar estes autómatos em Bases de Dados o que é muito comum, pois estes autómatos permitem representar o “ciclo de vida” de, por exemplo, um documento, um livro da biblioteca, uma tarefa, um meio complementar de diagnóstico médico, etc.

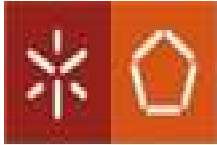




Ramificação condicional em função
do valor de uma expressão
(pseudostado de **escolha**)

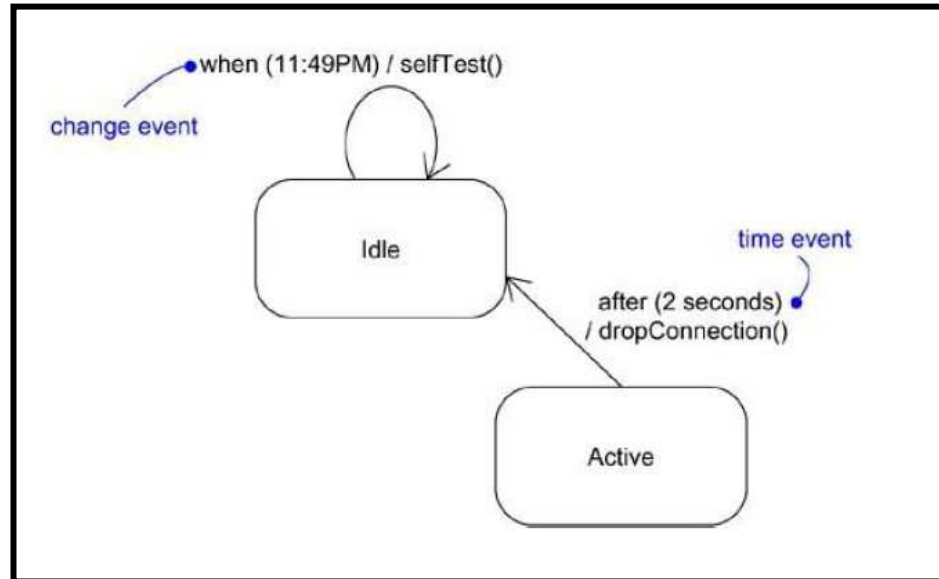




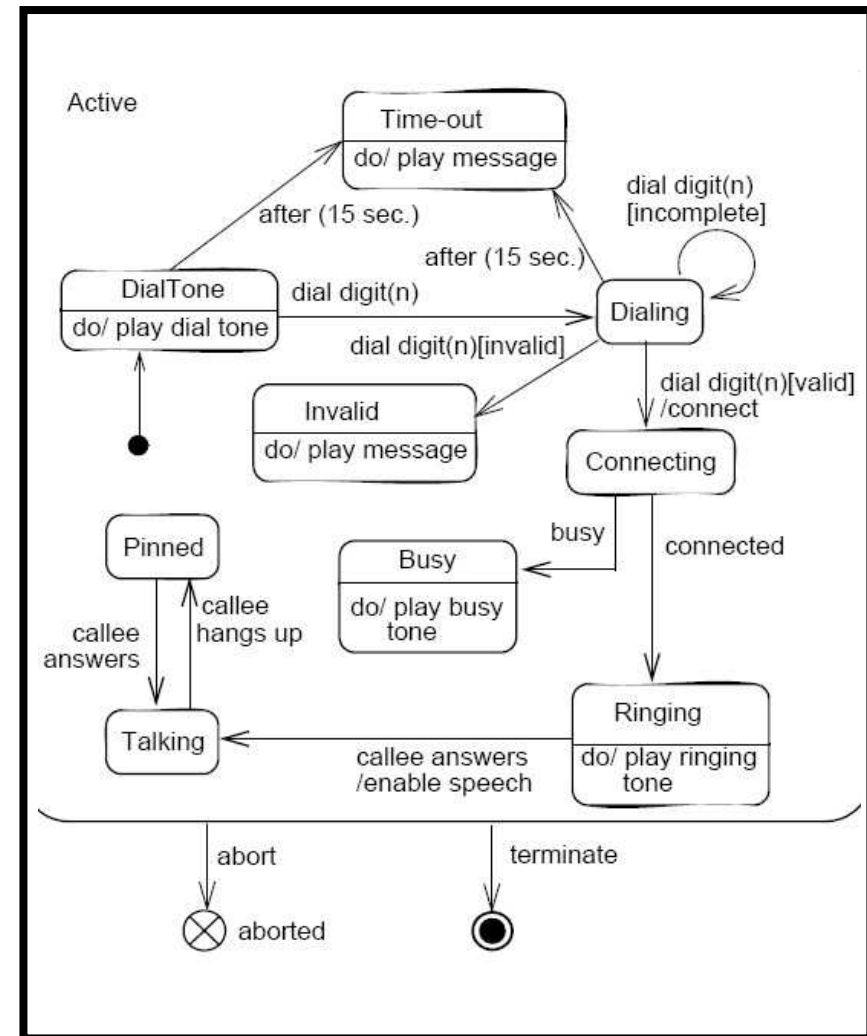


OUTROS EVENTOS

Tipo de Evento	Descrição	Sintaxe
<i>de tempo relativo ou de tempo absoluto</i>	Tempo relativo à entrada no estado origem ou tempo absoluto	<i>after(time)</i> <i>after(10s)</i> <i>after(21:00)</i>
<i>de tempo absoluto</i>	Chegada a um dado tempo absoluto; Verificação de condição	<i>when(time)</i> <i>when(expr_bool)</i> <i>when(1:00 AM)</i> <i>when(qt >= 50 ct)</i>
<i>de chamada de operação</i>	Recepção de uma chamada síncrona	<i>op(a:T)</i>
<i>de recepção de sinal</i>	Recepção de uma comunicação assíncrona	<i>sinal(a:T)</i>

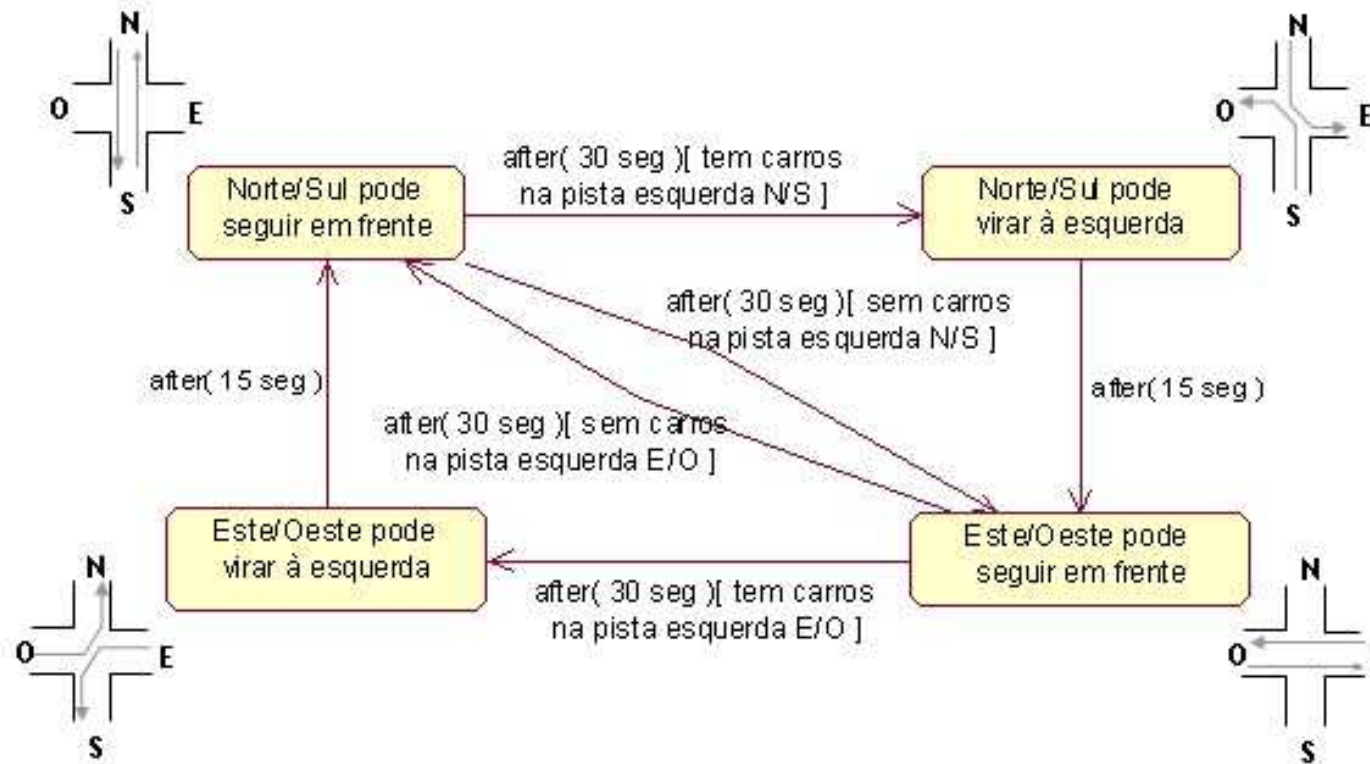


Exemplos de utilização dos eventos especiais **when()** e **after()**





Exemplo: Semáforo

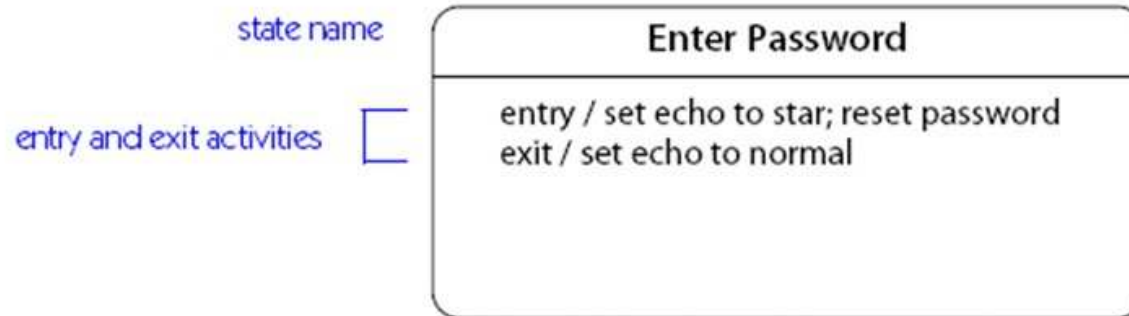


© Prof. João Pascoal, FEUP, 2001

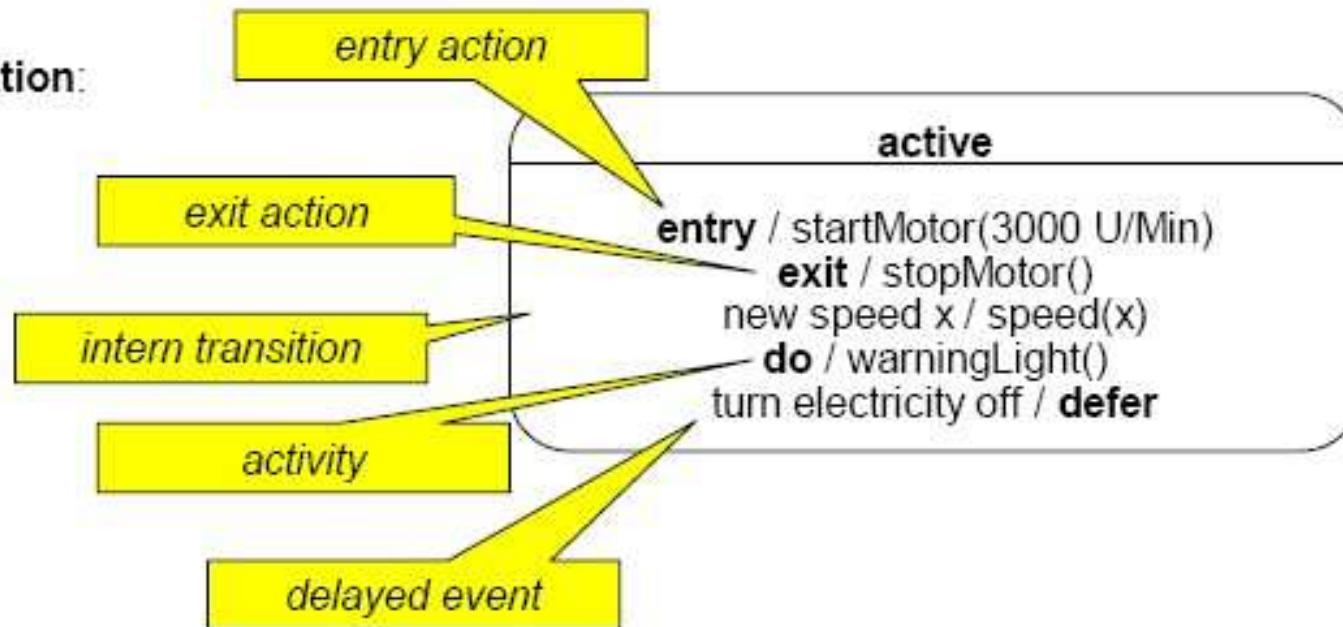


☐ Porém, os **estados** podem ter **actividades internas**, ou seja, quer quando são alcançados, ou atingidos, podem **executar de imediato as acções associadas ao evento interno entry**, quer quando são abandonados podem executar as acções associadas ao evento **exit**. Podem ainda **definir eventos que têm tratamento dentro do mesmo estado**, ou seja, que não provocam transições de estado mas apenas acções realizadas internamente e apenas com efeito interno (designadas “**self-transitions**”).

Evento/Acção	Definição
entry / acção	Acção executada ao entrar no estado
do / acção	Acção executada enquanto estado activo
exit / acção	Acção executada ao sair do estado
event / acção	Acção instantaneamente executada se evento ocorrer

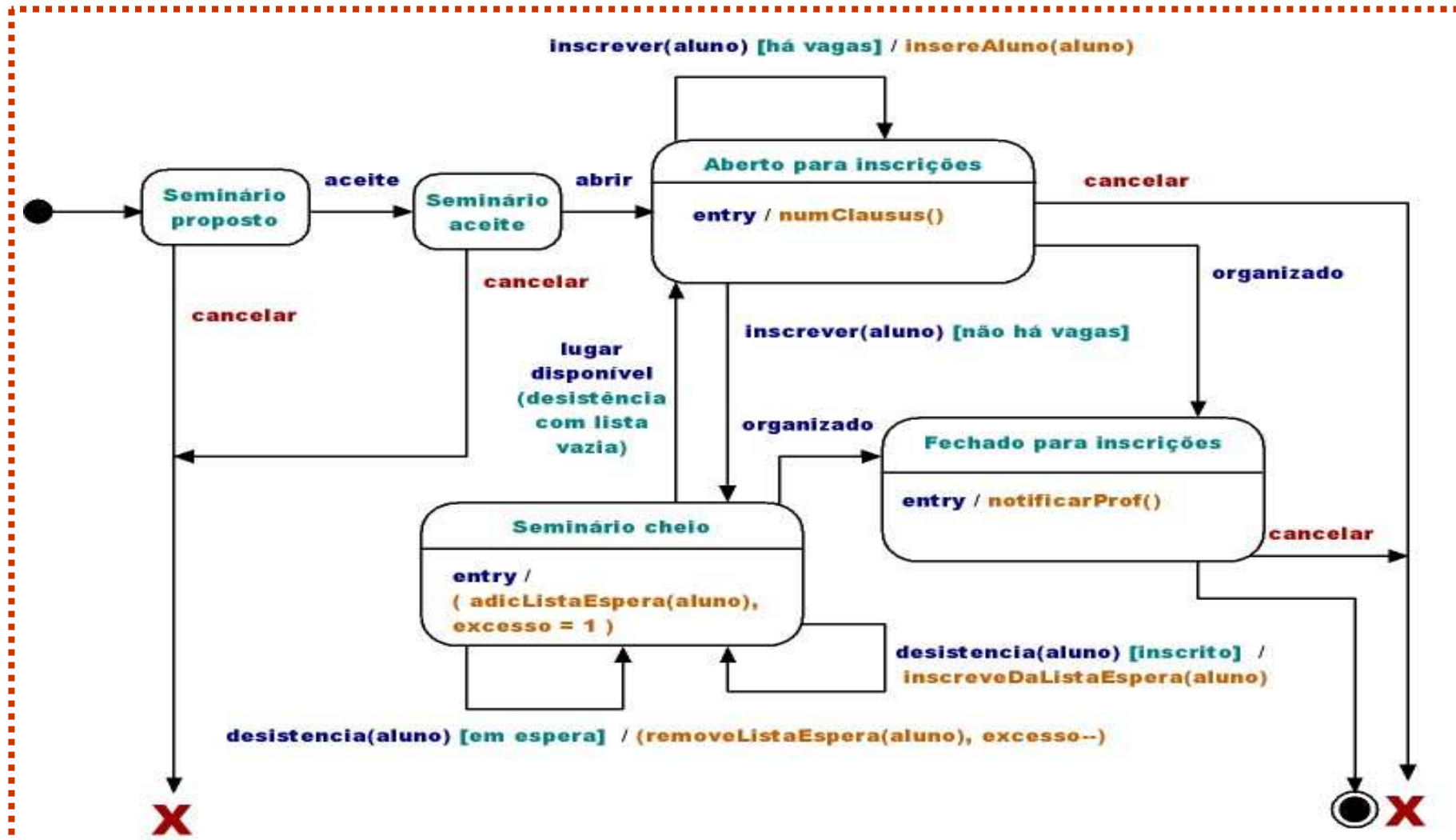


Notation:



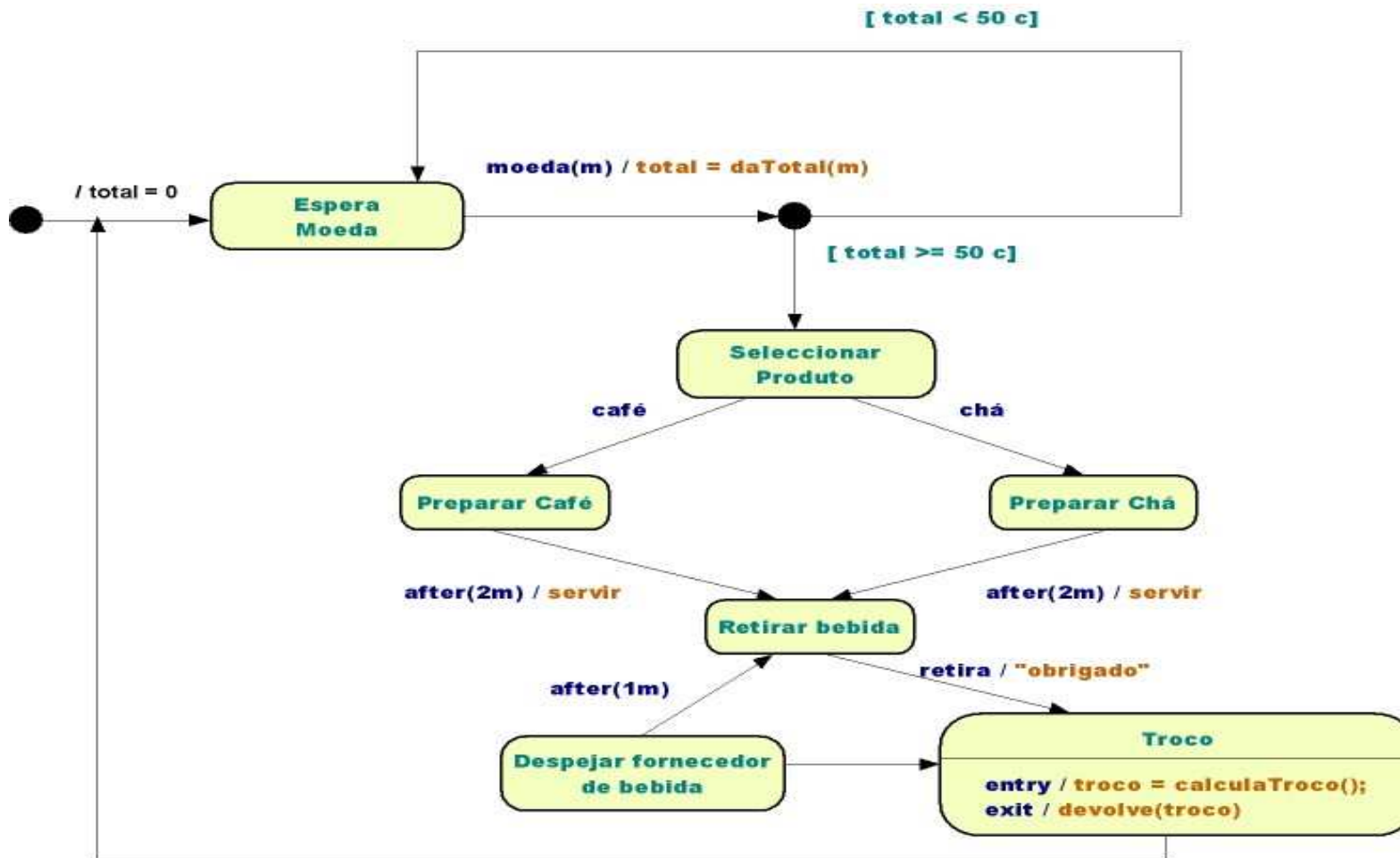


Inscrição de alunos num seminário com nº limitado de vagas



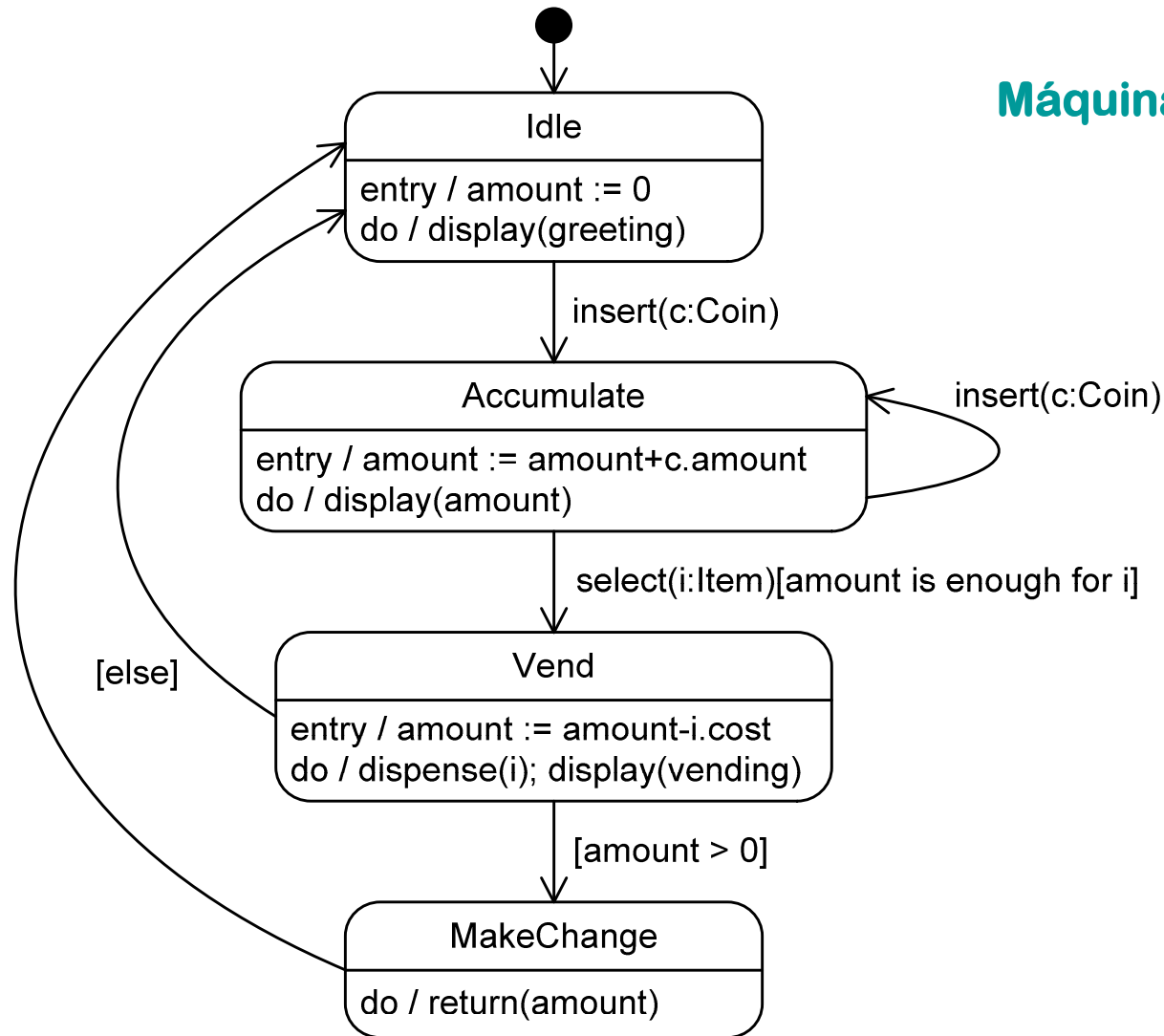


Máquina de Venda I



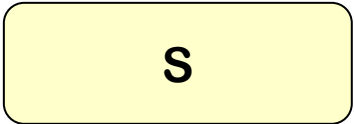
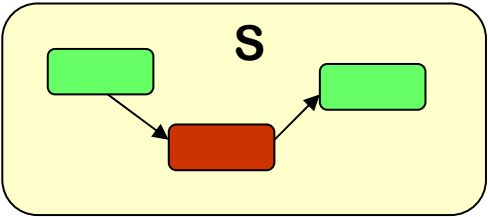
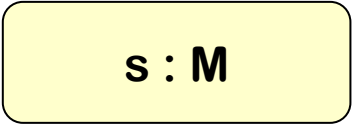
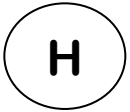


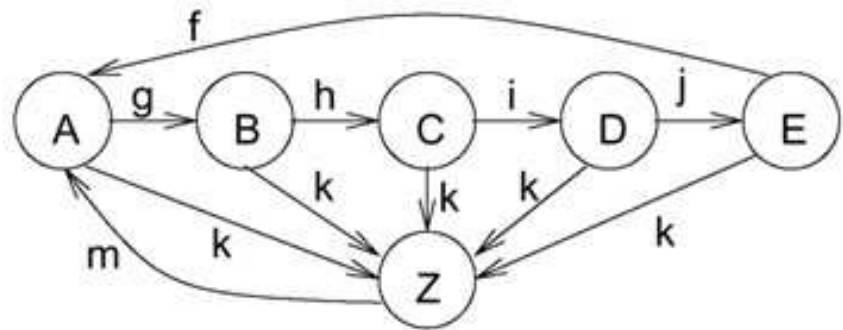
Máquina de Venda II



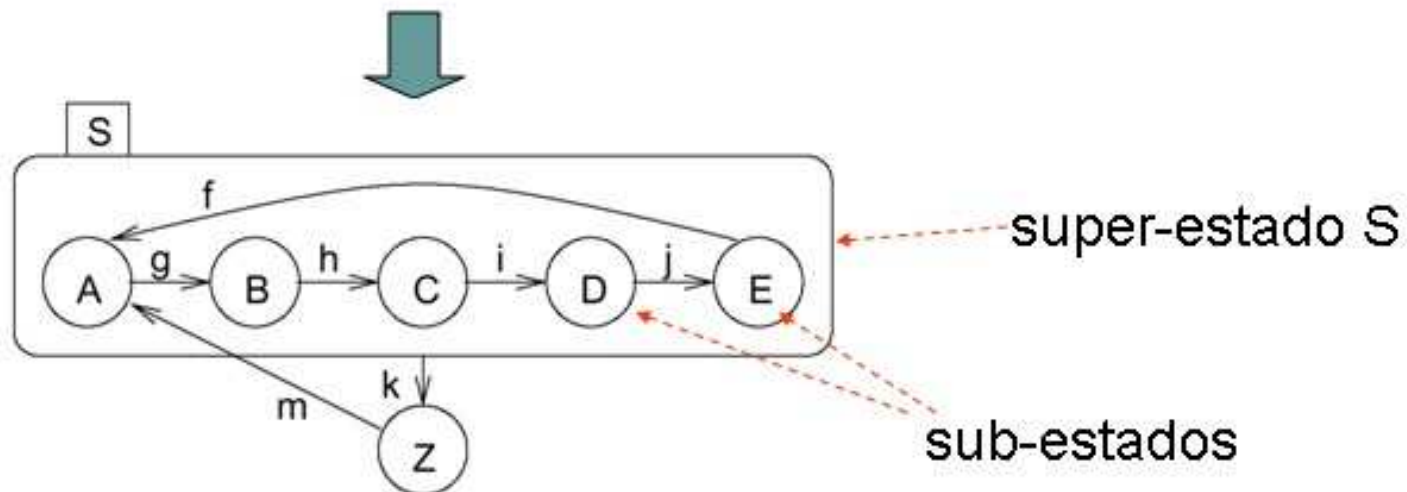


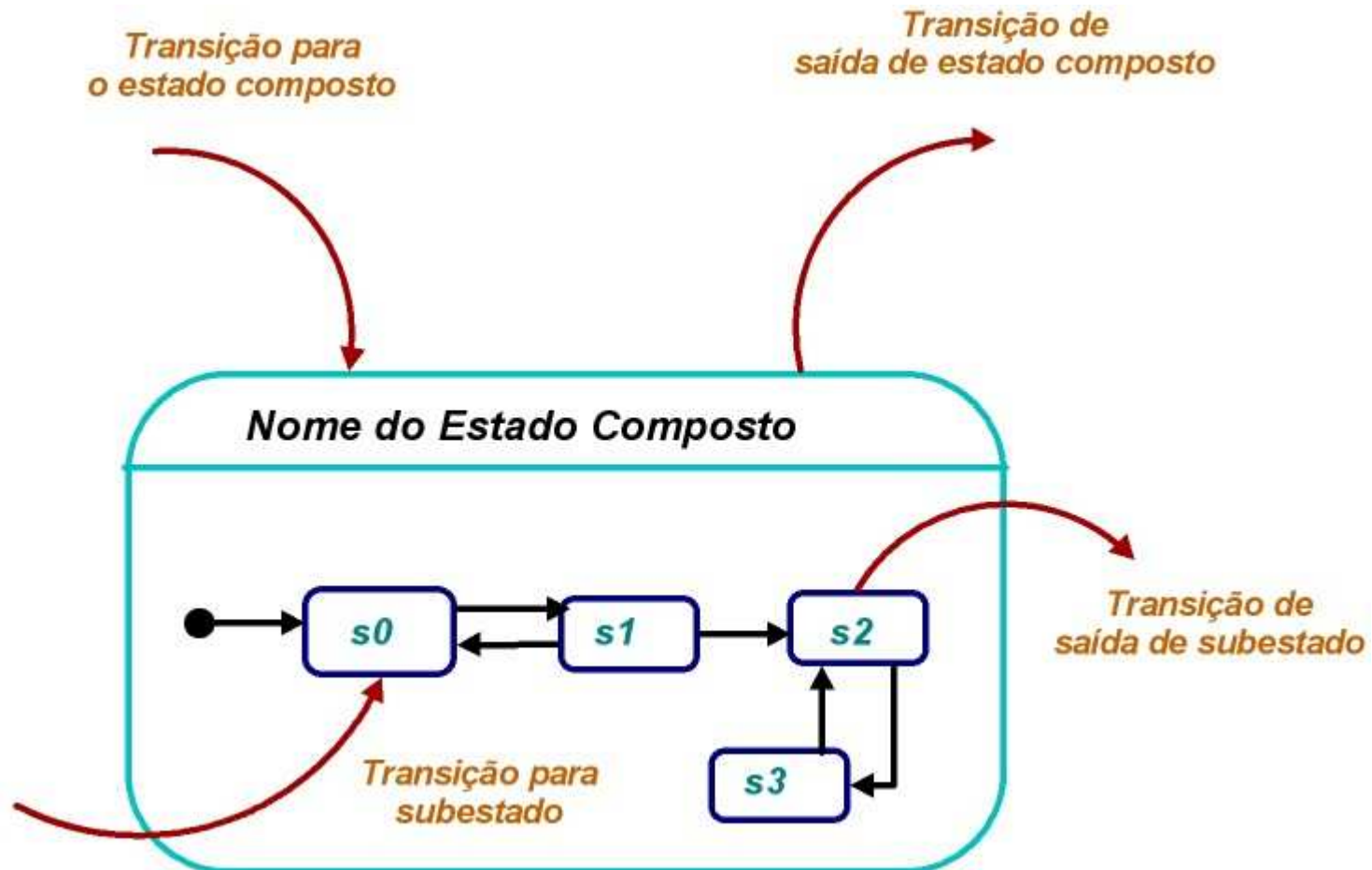
OUTROS ESTADOS I

Tipo de Estado	Descrição	Notação
<i>simples</i>	Estado sem subestrutura	
<i>composto sequencial</i>	Estado composto formado por subestados, dos quais apenas um está activo quando o estado composto está activo	
<i>estado de submáquina</i>	Estado que referencia uma máquina de estados	
<i>estado história</i>	pseudo-estado cuja activação repõe o estado previamente activo num estado composto	



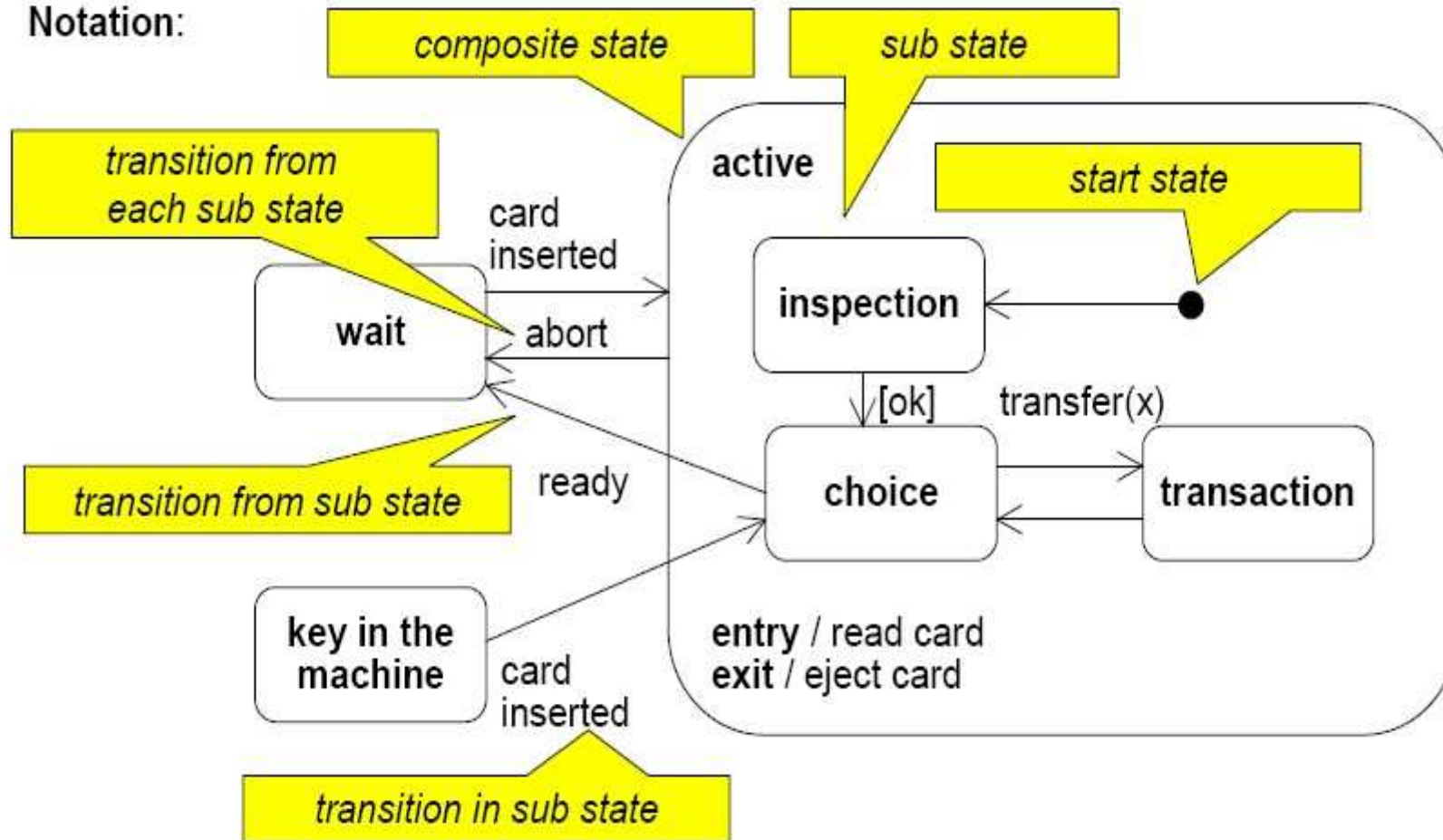
Estados compostos são muito úteis na estruturação dos diagramas de estado

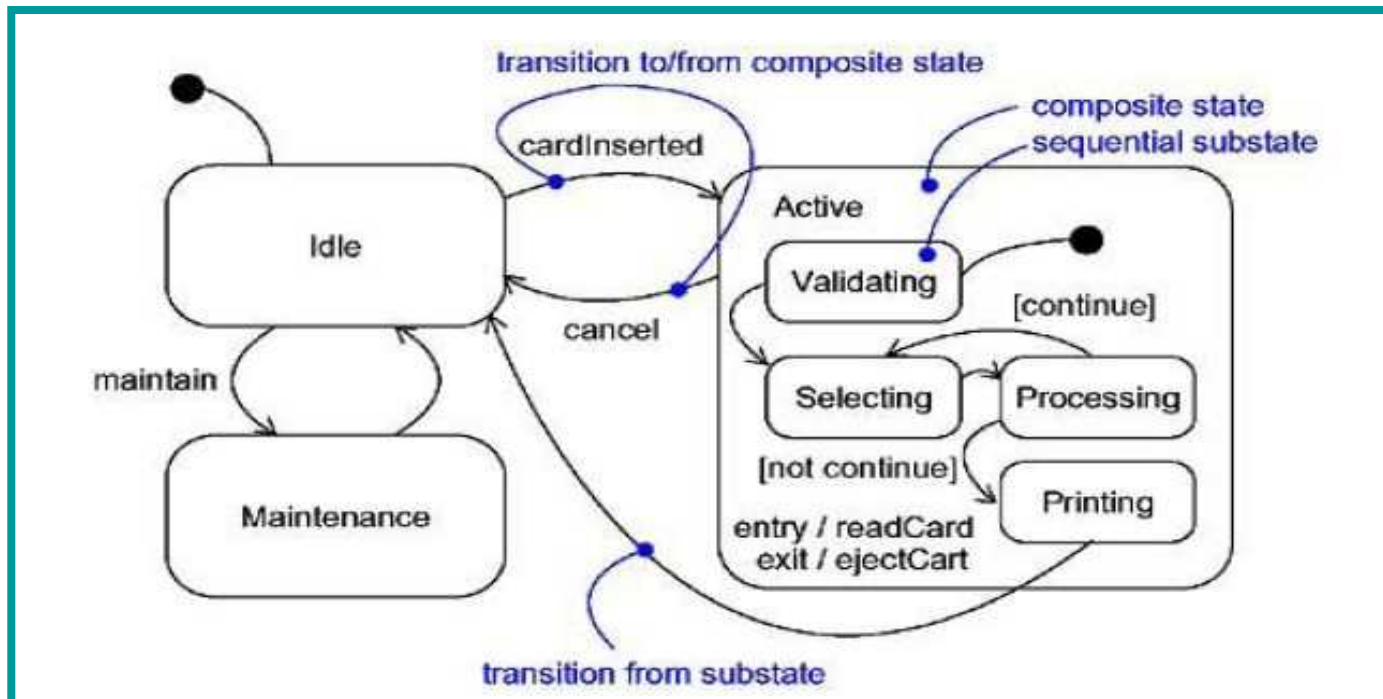
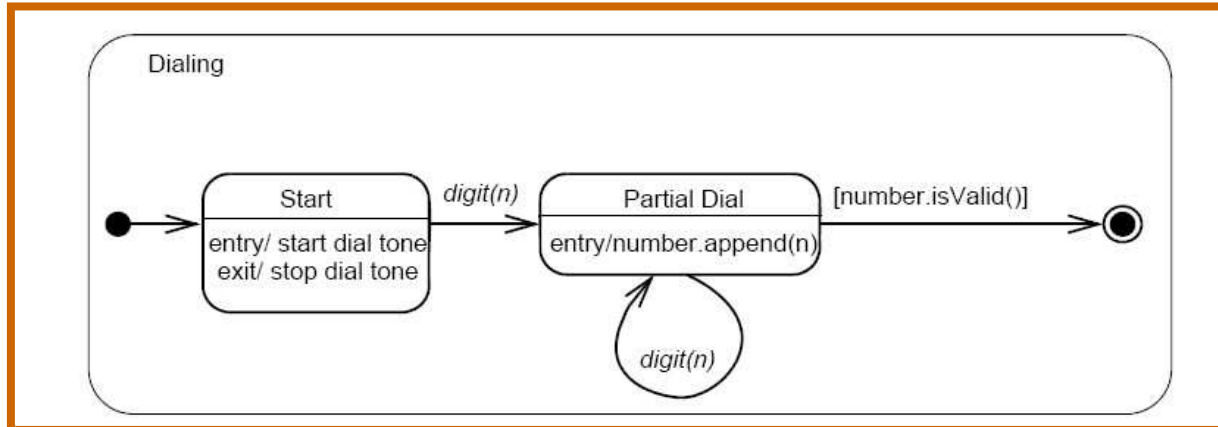


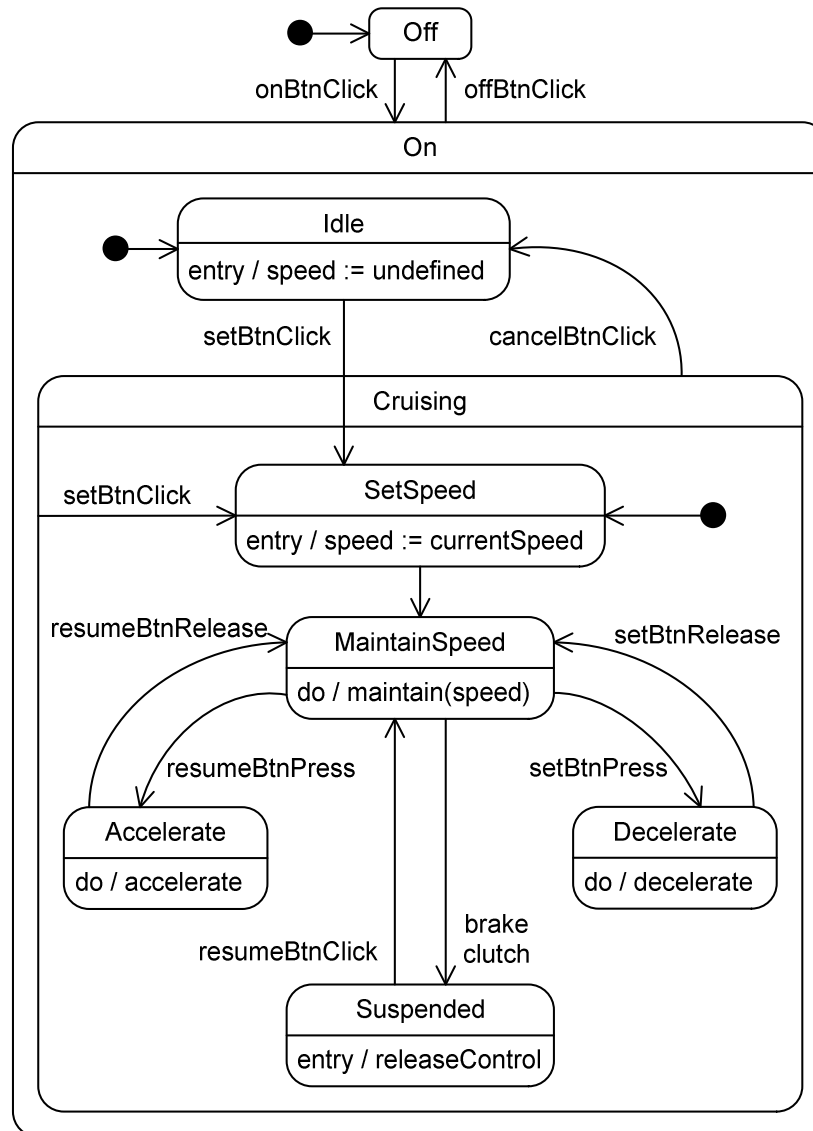




Notation:





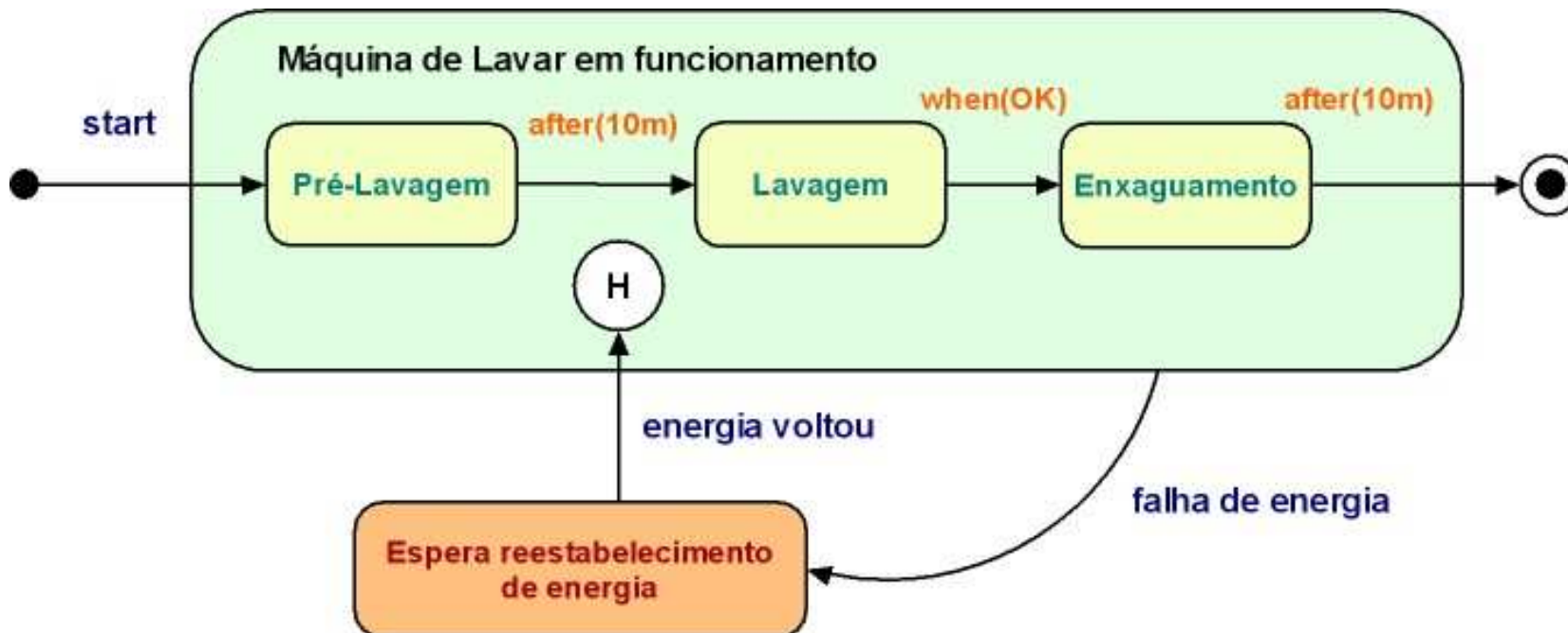


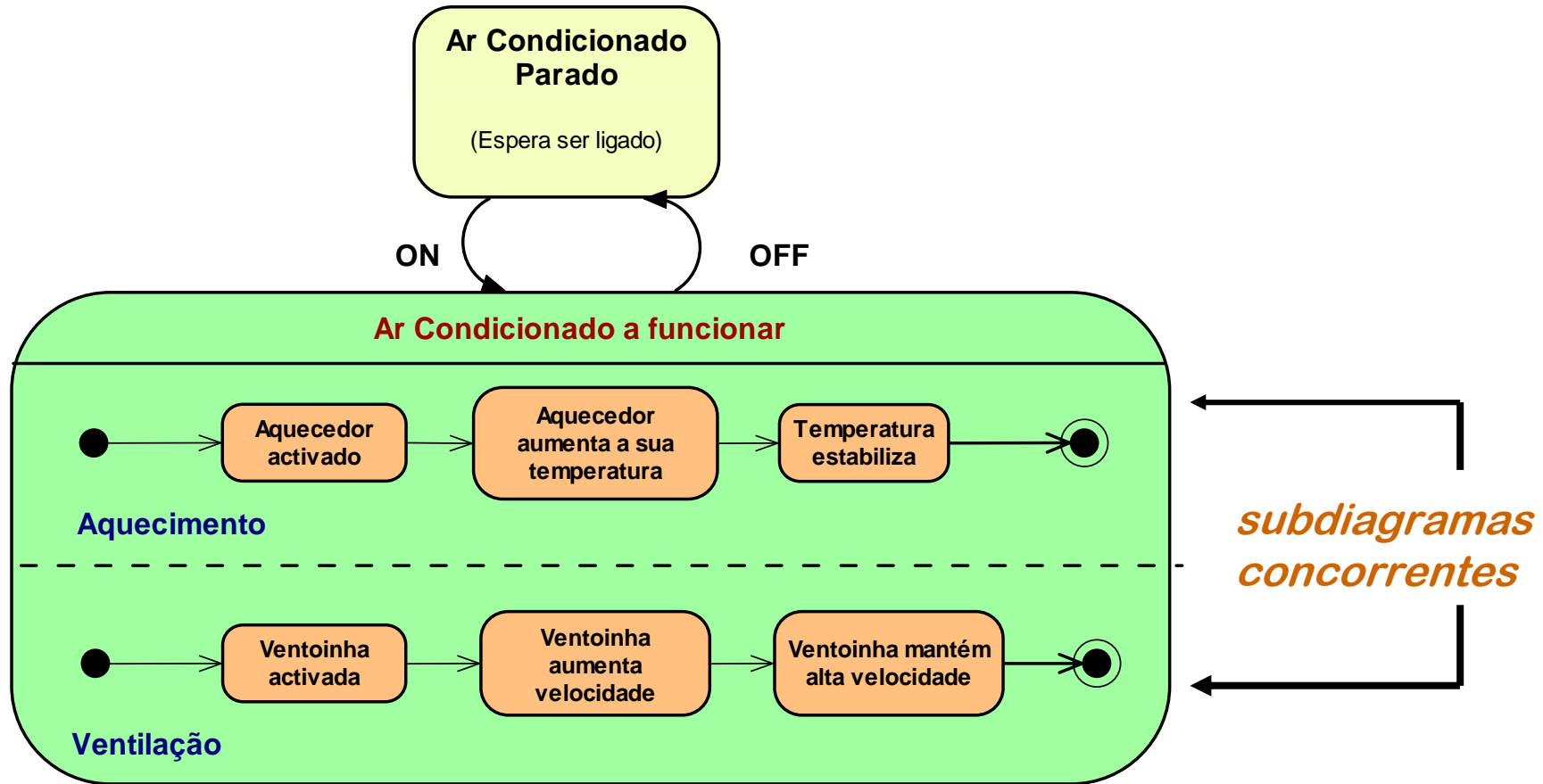
Note-se que o estado **On** é composto por dois subestados, **Idle** e **Cruising**, sendo **Cruising** igualmente um estado composto.

O nível de aninhamento é ilimitado, ou melhor, apenas limitado pela capacidade de compreensão e clareza.



☐ O **pseudo-estado de memória** indica que a actividade da máquina é retomada no exacto **último estado completo** em que esta se encontrava aquando da última saída (cf. **hibernar** em PCs).



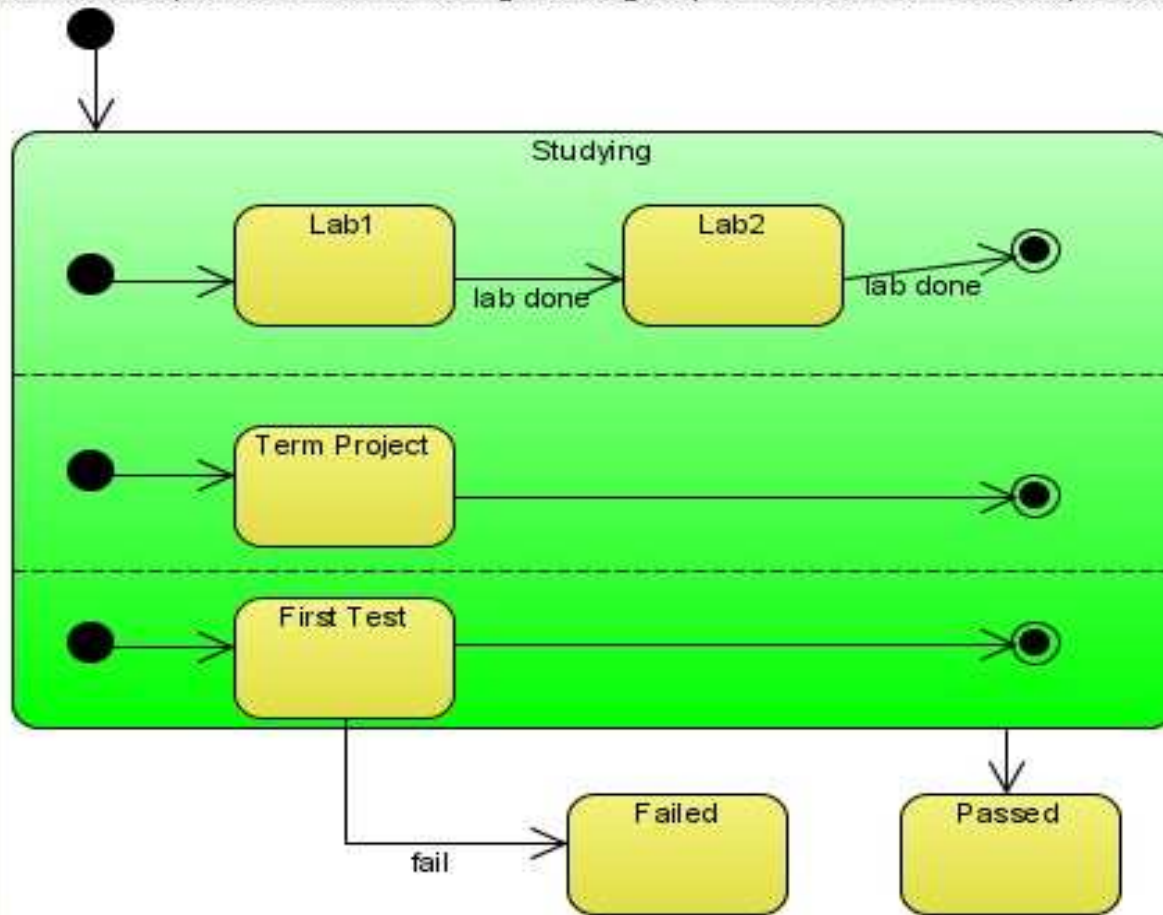


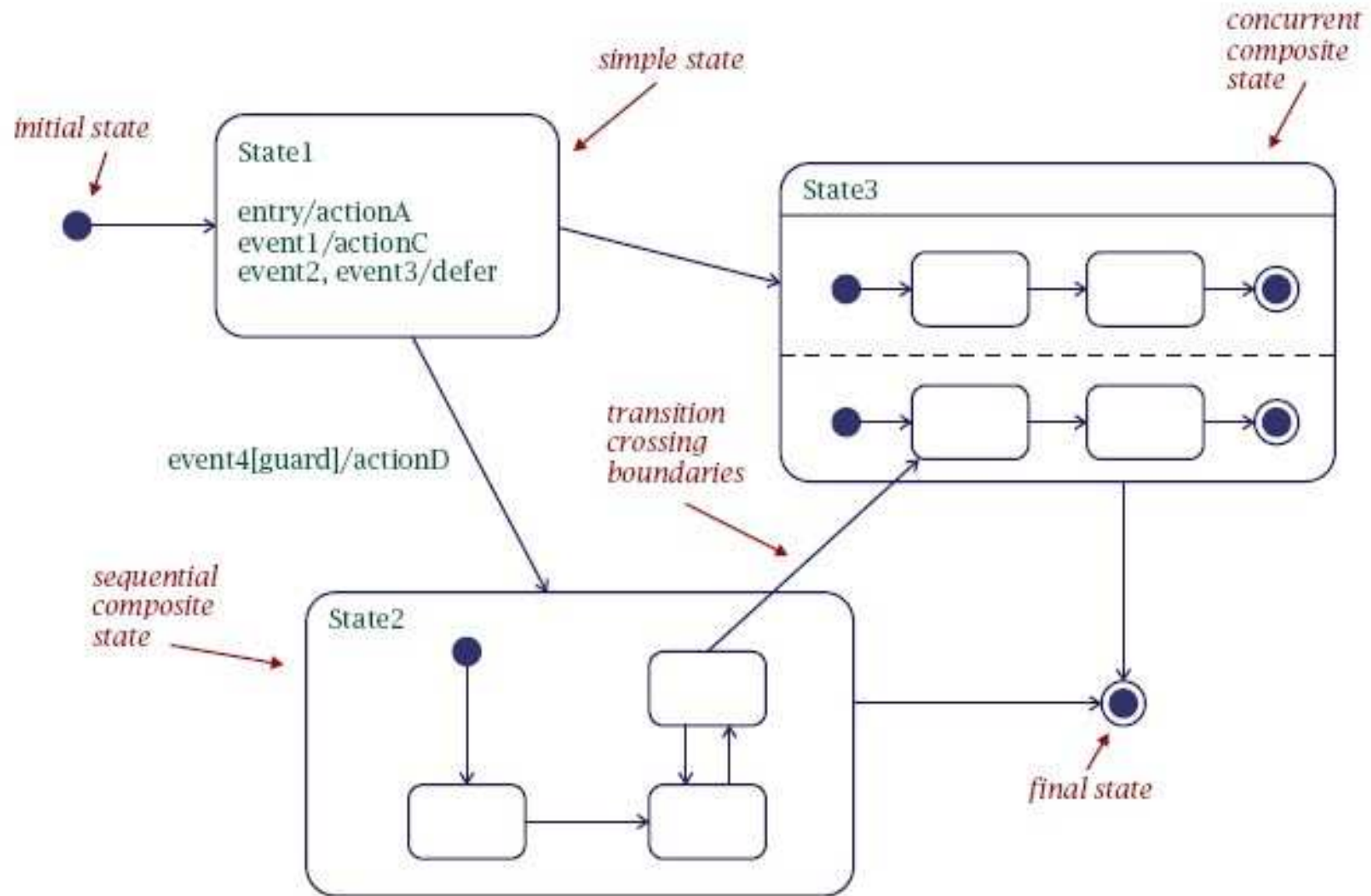
Quando se entra neste estado, os dois subdiagramas são executados de forma concorrente. O comportamento termina quando terminarem os 2.



Region

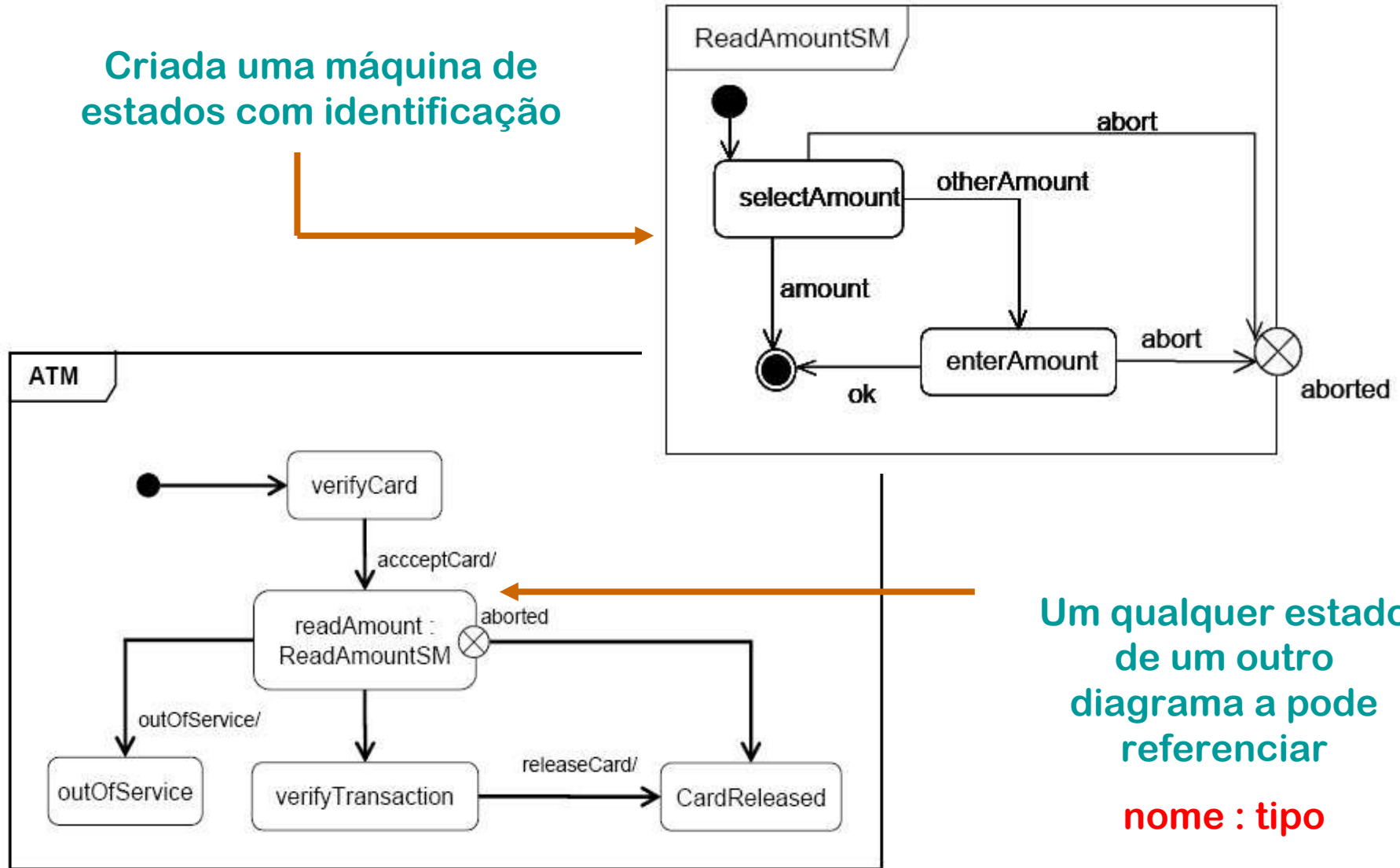
A region is an orthogonal part of either a composite state or a state machine. It contains states and transitions. (OMG Unified Modeling Language Specification - UML 2.0 Superstructure Specification, p. 597)







Criada uma máquina de estados com identificação

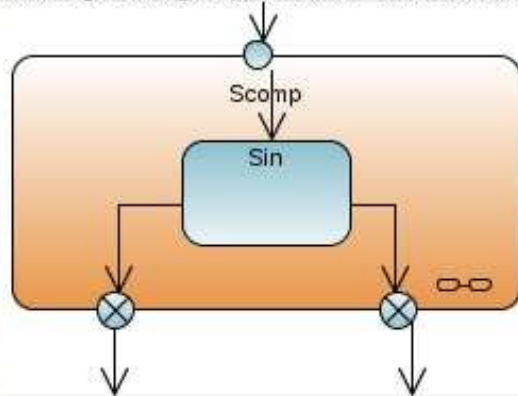


Um qualquer estado de um outro diagrama a pode referenciar
nome : tipo



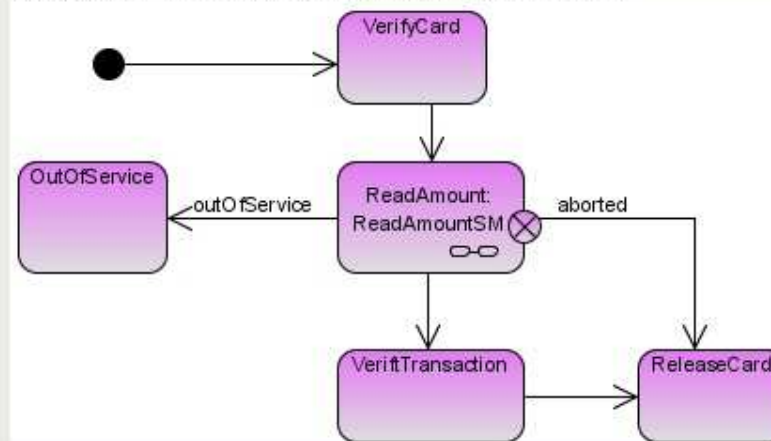
Entry point

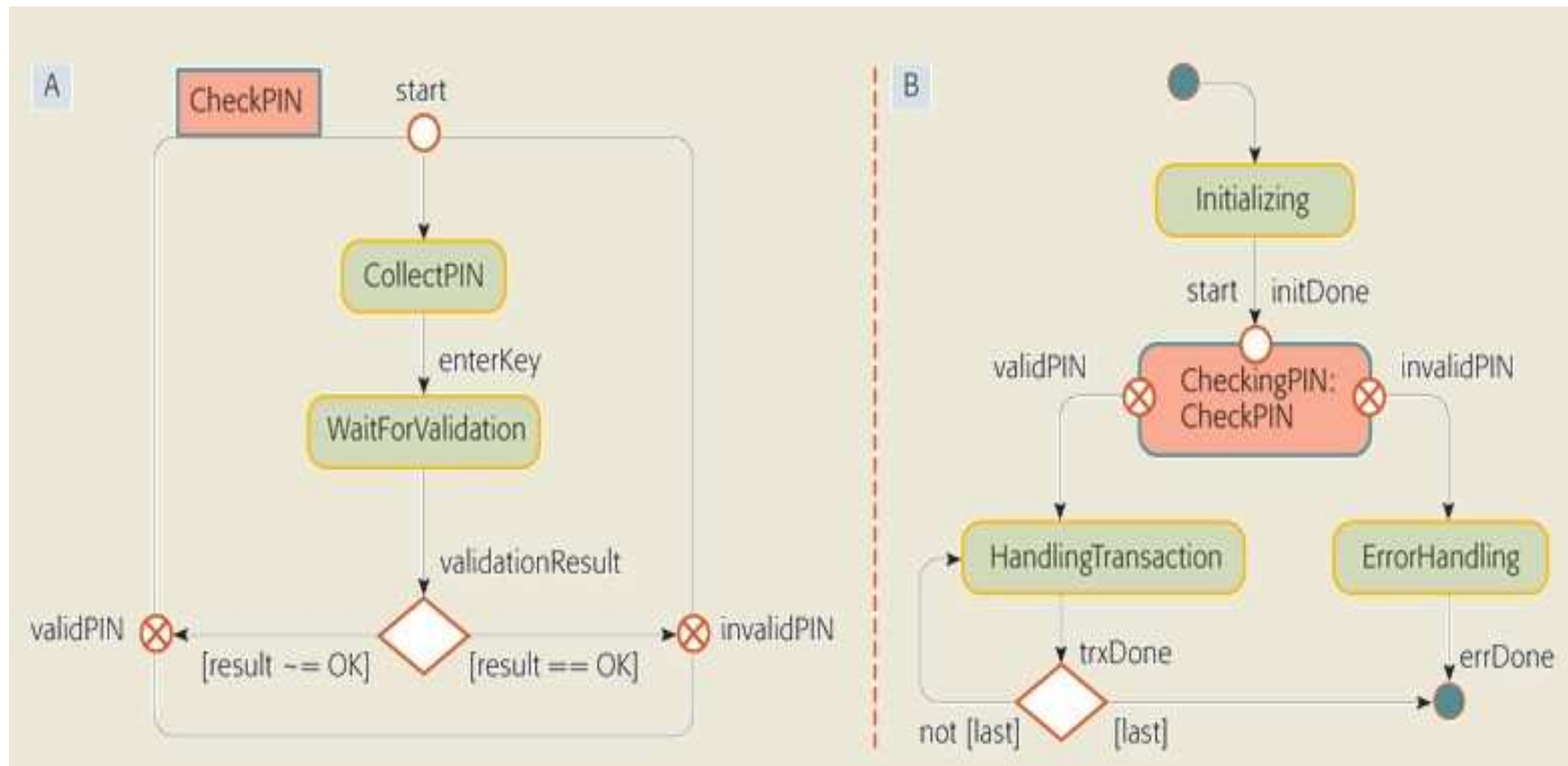
An entry point pseudostate is an entry point of a state machine or composite state. In each region of the state machine or composite state it has a single transition to a vertex within the same region. (OMG Unified Modeling Language Specification - UML 2.0 Superstructure Specification, p. 592)



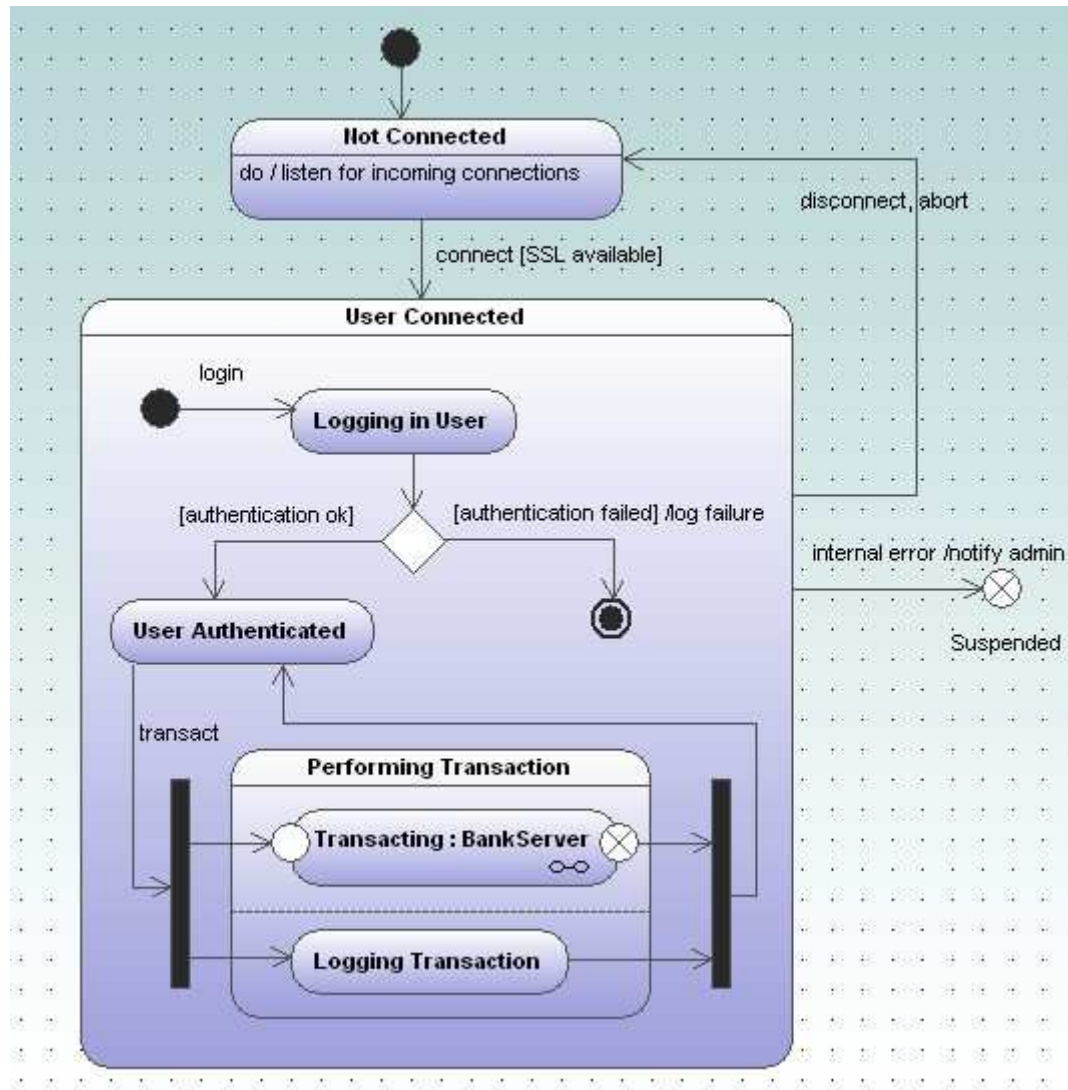
Exit point

An exit point pseudostate is an exit point of a state machine or composite state. Entering an exit point within any region of the composite state or state machine referenced by a submachine state implies the exit of this composite state or submachine state and the triggering of the transition that has this exit point as source in the state machine enclosing the submachine or composite state. (OMG Unified Modeling Language Specification - UML 2.0 Superstructure Specification, p. 592)





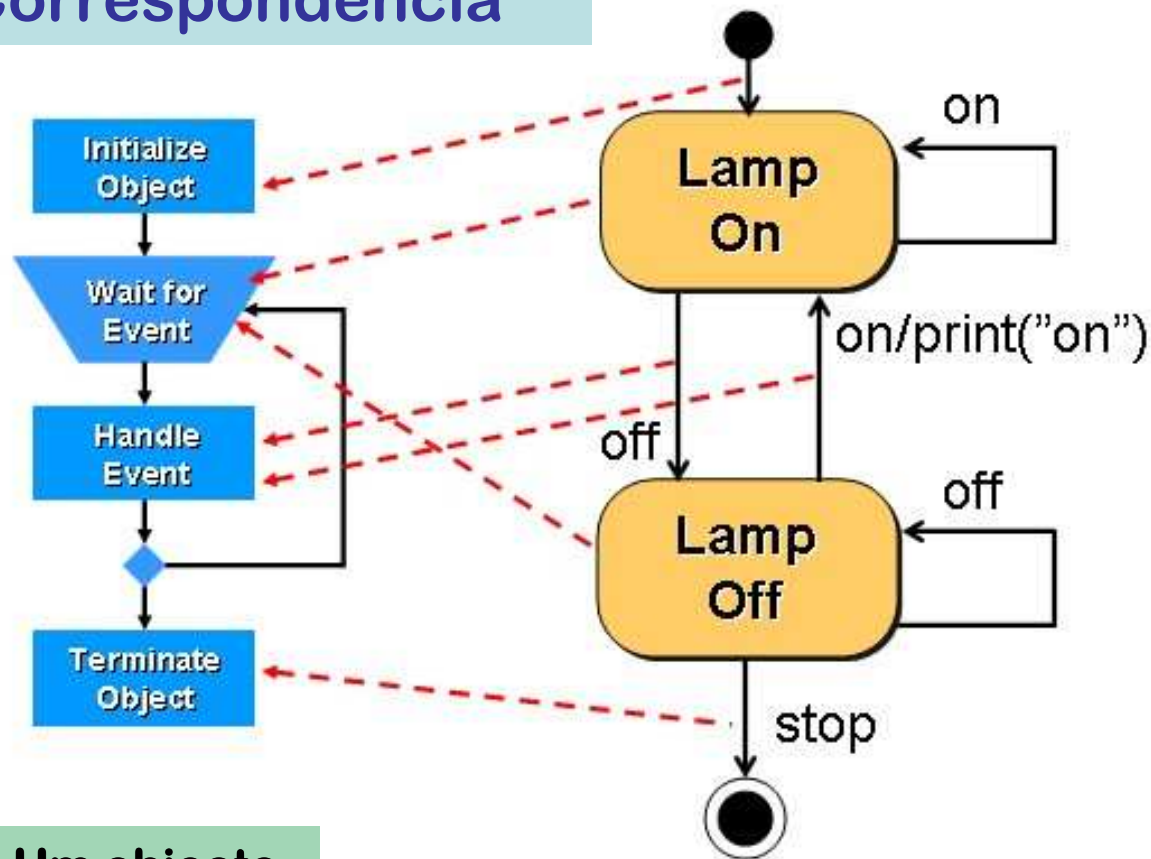
☐ Submáquina **CheckPIN**



Especificação dos passos do estabelecimento de uma ligação segura (SSL) a um servidor bancário para realizar transacções



Correspondência

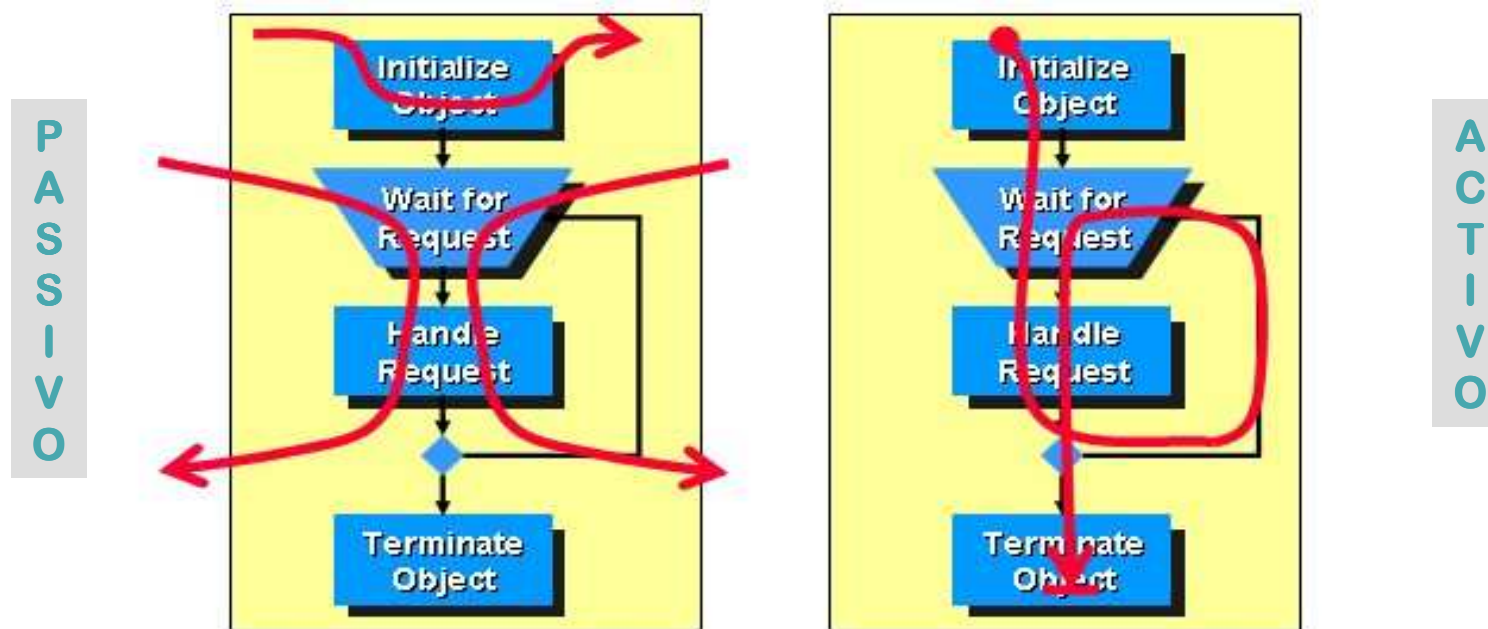


Um objecto servidor

Uma FSM



- ▶ **Objectos passivos** possuem fluxos de execução que dependem de “energia” e “inteligência ou controlo” exterior (cf. **invocação de métodos**);
- ▶ **Objectos activos** possuem “energia” para definir e executar os seus próprios fluxos de execução (“threads”, cf. **processos leves**);

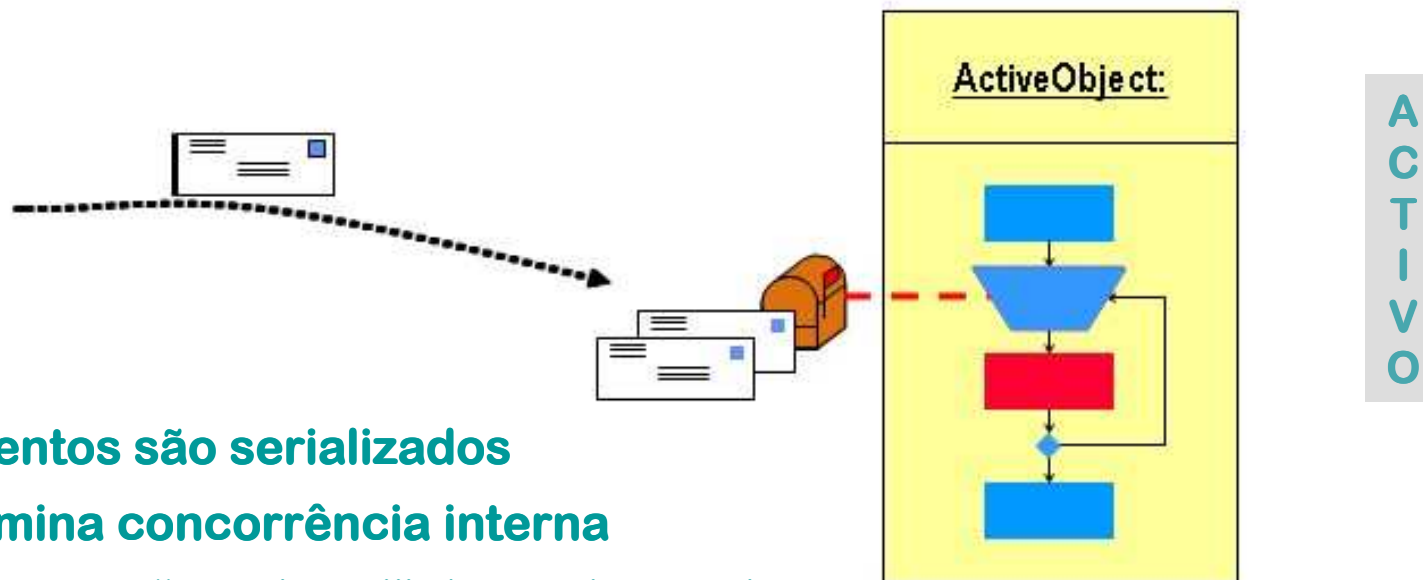


▣ Semântica de “run-to-completion” => tratar 1 evento de cada vez

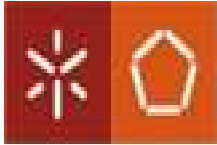


▶ **Objectos activos** possuem “energia” para definir e executar os seus próprios fluxos de execução (“threads”, cf. **processos leves**);

▣ Semântica de “**run-to-completion**” => tratar 1 evento de cada vez



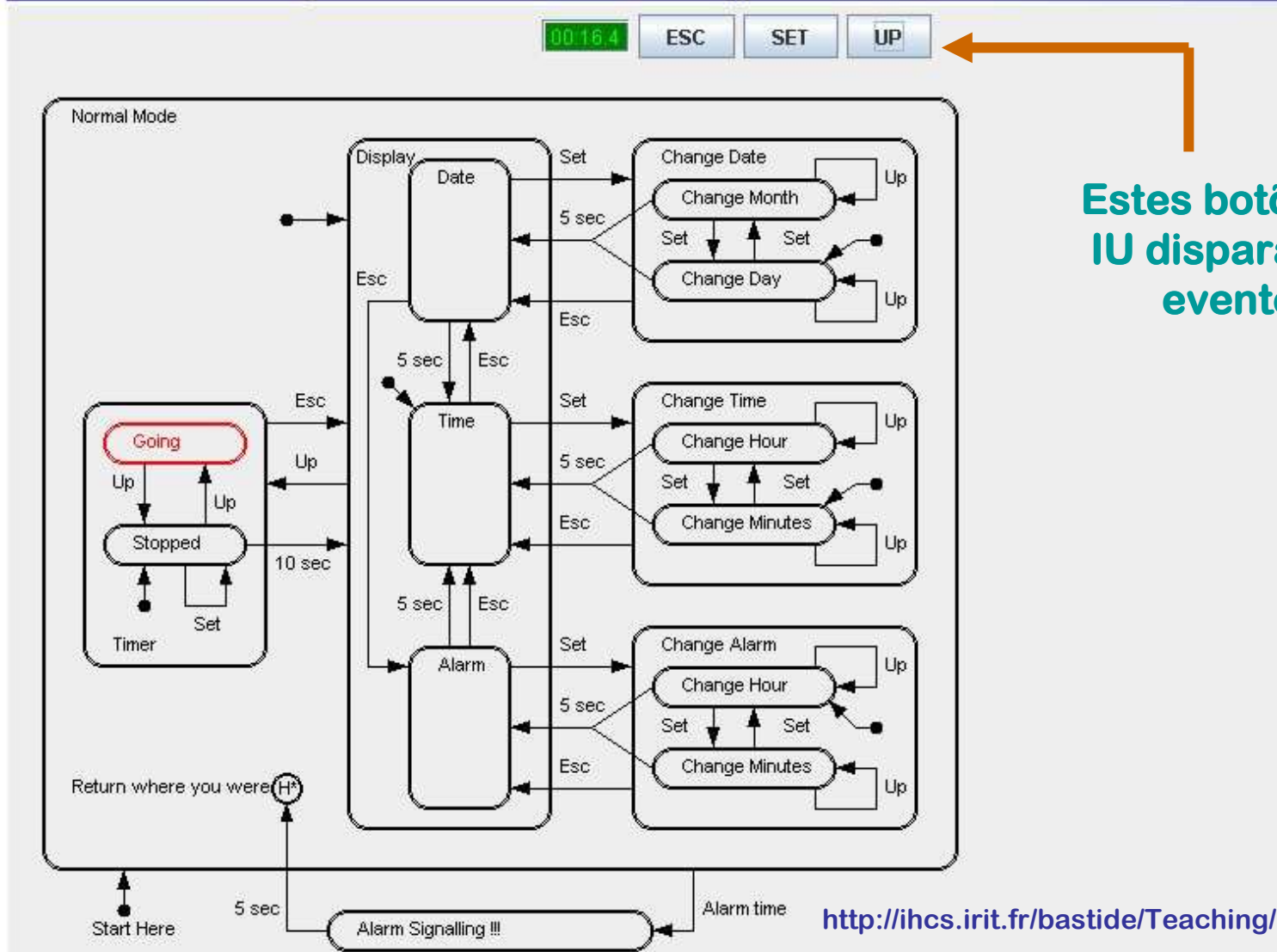
- ▶ **Eventos são serializados**
- ▶ **Elimina concorrência interna**
- ▶ **Minimiza “overhead” de mudança de contexto**



- ▣ **Diagramas de Estado** permitem-nos descrever o comportamento de uma entidade importante do sistema de forma completa, ou seja, trazendo para um único diagrama o comportamento que em geral está especificado de forma dispersa em vários UC ou DS.
- ▣ Os **diagramas de actividade** também permitem uma visão mais sistémica, pois permitem especificar fluxos importantes de actividades que envolvem vários objectos, use cases e até actores.
- ▣ **Diagramas de Estado** não são adequados para descrever ou analisar colaborações entre entidades/objectos.
- ▣ **Diagramas de Estado** não são usados para descrever todas as classes do sistema, mas aquelas que exibam comportamento interessante ou complexo. Alguns autores usam DMEs para especificar certas classes relacionadas com a Interface com Utilizador.



A digital clock with its control logic modelled as a UML StateChart

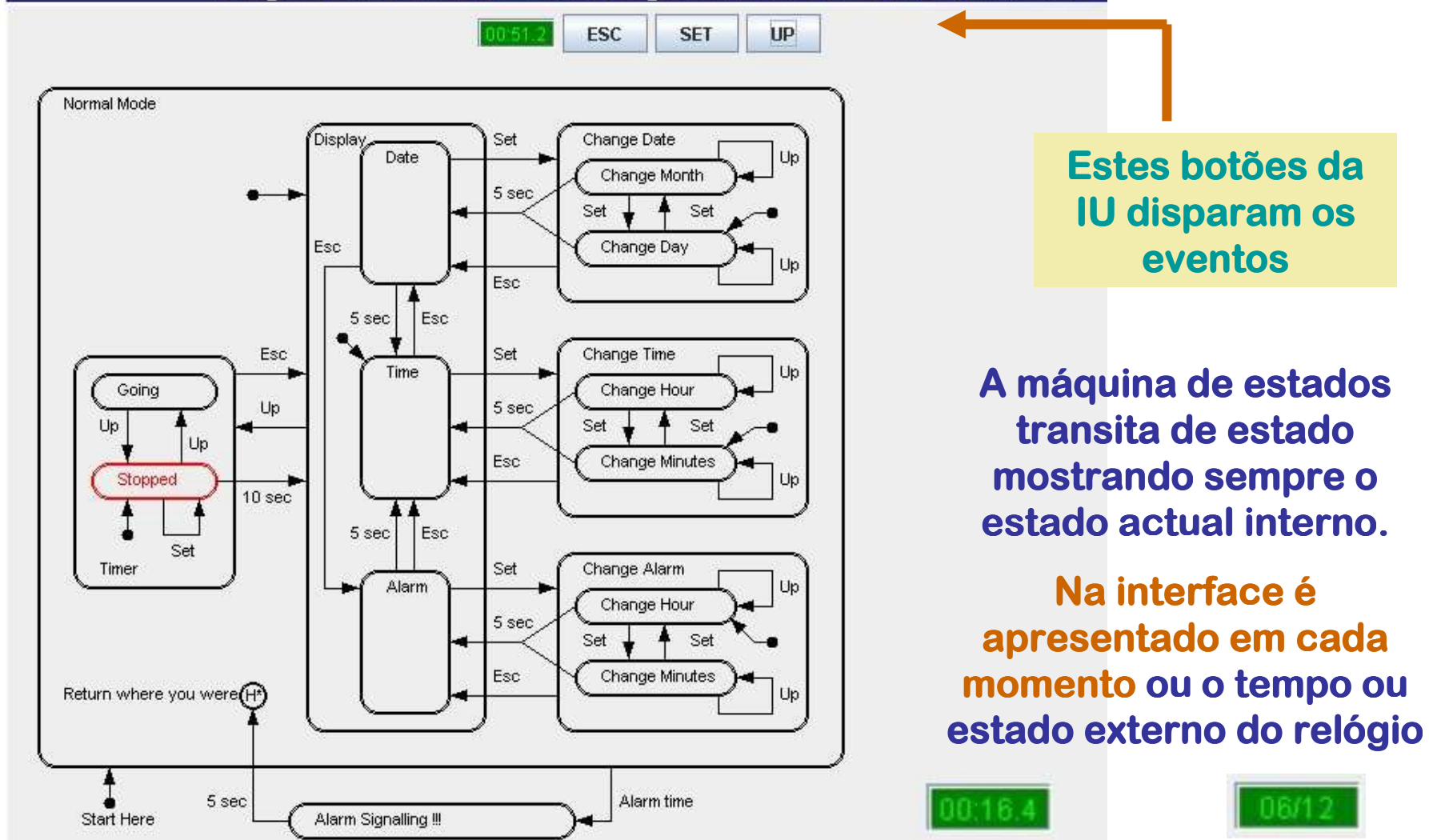


Estes botões da IU disparam os eventos

<http://ihcs.irit.fr/bastide/Teaching/UML/StateCharts/>



A digital clock with its control logic modelled as a UML StateChart



Estes botões da IU disparam os eventos

A máquina de estados transita de estado mostrando sempre o estado actual interno.

Na interface é apresentado em cada momento ou o tempo ou estado externo do relógio

