

Essentials of state and activity diagrams - Chap 11 of text

Essentials of state and activity diagrams - Chap 11 of text	1
• A few general points	2
• State diagrams.....	2
Simple example, from library case study.....	2
Level of abstraction (section 11.1.2)	3
States, transitions, events (section 11.1.3)	3
Actions (section 11.1.4).....	4
General	4
Case where action is in response to a specific event	4
Case where action on entry to a state is same for all transitions (in)	5
Case where action on exit from a state is same for all transitions (out).....	5
Guards (section 11.1.5).....	6
On-line shopping (ex Software Design by E. Braude, Wiley, 2004).....	7
Example of use of states (called modes) in an actual project	8
Some other points on state diagrams:	13
• Activity Diagrams.....	13
General	13
Example of a “business level” activity diagram	14
Summary of activity chart notation	15
Note2: Web-ref. to Nassi-schneiderman diagrams & Flow Charts	15

• A few general points

- Concerned with showing how an object decides to react to a message
- In chapter 11 concern is with the commonest use of state (or "statechart" diagrams) namely "show how an object reacts to receiving a message by sending messages"
- Activity diagrams are (in a sense) a variant of state diagrams that *can* aid understanding of complex activities. Sometimes may be better to use interaction (see earlier) rather than activity diagrams.

• State diagrams

Simple example, from library case study

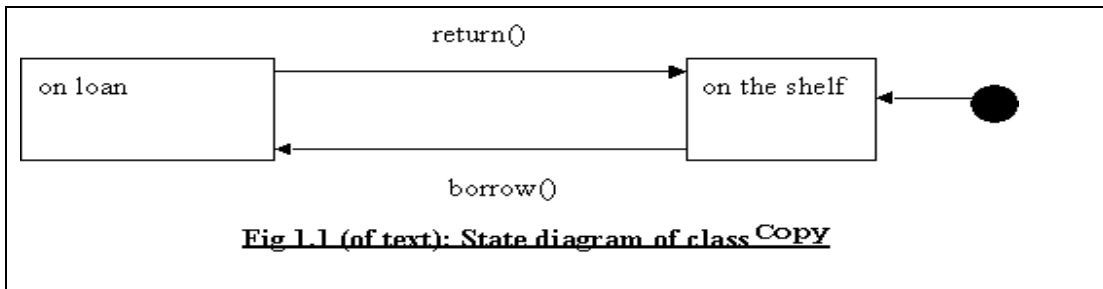


Fig 1.1 (of text): State diagram of class Copy

- It may be decided that the class Copy should have an attribute onShelf (true/false if copy is/is not in library)
- Messages borrow() and return() are *events* that cause state *transitions*
- Message borrow() informs Copy object it has just been

Black "blob" is an optional *start marker*. Useful if objects of a class always start in the same state

borrowed. It *SHOULD* only arrive if onShelf = true but what happens otherwise?

- [Practical suggestion-departure from UML which "ignores"] A common solution to the situation that an event arrives when an object is in the "wrong" state is "to have a single, globally accessible object of a class Error, whose sole responsibility is to report errors. An object that receives a message it was not expecting sends a message to the error object describing what happened. This error handling is not shown on the state diagram to avoid clutter. [Implications for coupling are ??? In Java, note "throw" and "catch" for exception handling]

Level of abstraction (section 11.1.2)

In general, the values of *all* the attributes of an object define its state. In fact, as noted in the text, to be complete one should say that the object's state also depends on the state of objects it is linked to etc. However, in a state diagram, we are often concerned with a particular aspect of an object which is defined by just a few of its attributes (e.g. onShelf) - for this purpose we are indifferent to the values of its other attributes (e.g. library number of copy, book it is a copy of) and do not depict them.

States, transitions, events (section 11.1.3)

For the record, the UML syntax (notation) is

states - shown as rounded boxes [*sharp corners in notes - apologies*]

transitions between states - shown as arrows

events that cause transitions - most common kind is receipt of a message

- shown by writing the message on the transition arrow

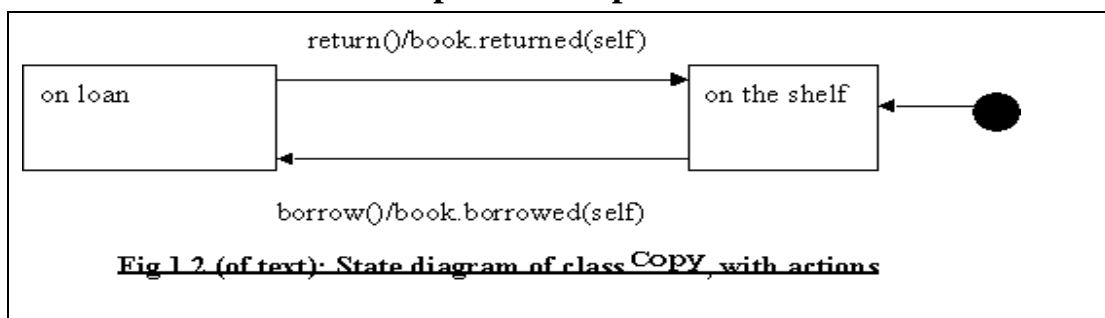
start marker - shown as black blob with unlabeled arrow to initial state

stop marker - shown as ringed black blob with unlabeled arrow to it. 

May be several or no stop markers present. Means object has reached end of its life & will be destroyed

Actions (section 11.1.4)**General**

An event is something done to an object, such as *it being sent* a message. An action is something that an object does, such as *it sending* a message.

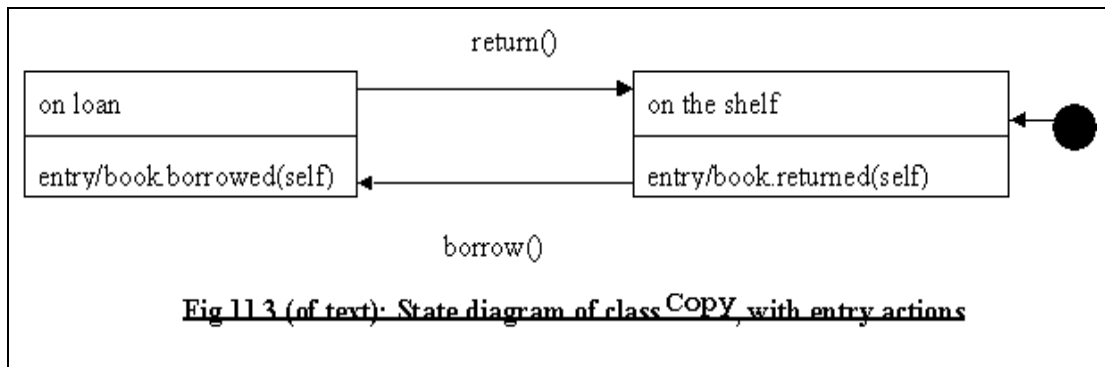
Case where action is in response to a specific event

Syntax (notation):

- Action is shown after the event on transition arrow, separated by "/"
- "book" identifies object to which message is being sent
- "returned(self)" is a message with a parameter. Here, parameter is an object of class Copy, namely (a reference to) the object itself.
(Similarly, for "/book.borrowed(self)")

Meaning:

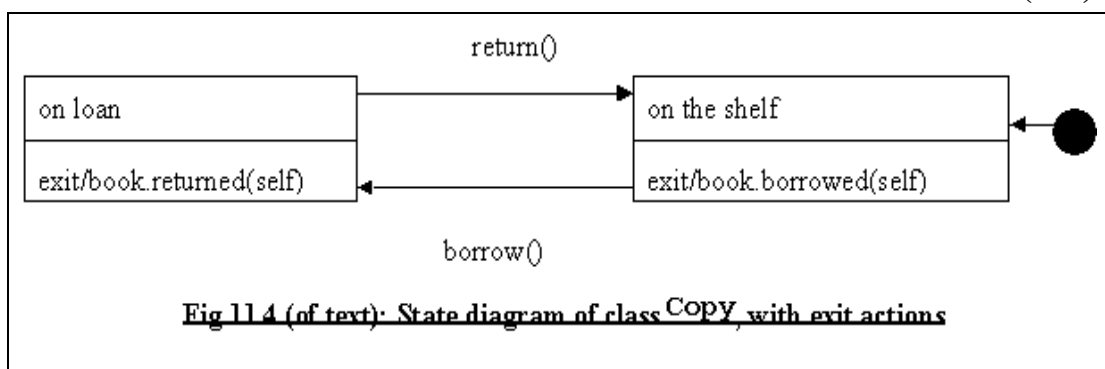
The Copy object sends messages `borrowed(self)` and `returned(self)` as part of its reactions to receiving the messages `borrow()` and `return()`, respectively.

Case where action on entry to a state is same for all transitions (in)

-- Every time an object enters a state there is an entry event. Normally, this is not shown explicitly unless, as here, actions are associated with it.

-- Action is shown inside the state, as a reaction to special event entry.

-- The Copy object sends messages `borrowed(self)` and `returned(self)` in reactions to entering the "on loan" and "on the shelf states", respectively.

Case where action on exit from a state is same for all transitions (out)

-- Every time an object leaves a state there is an exit event. Normally, this is not shown explicitly unless, as here, actions are associated with it.

-- Action is shown inside the state, as a reaction to special event exit.

-- Copy object sends `borrowed(self)` and `returned(self)` in reactions to exiting "on the shelf" and "on loan" states, respectively.

Remark: Alternative to above would be to put entry & exit in the 'on loan' state.

Guards (section 11.1.5)

We had before

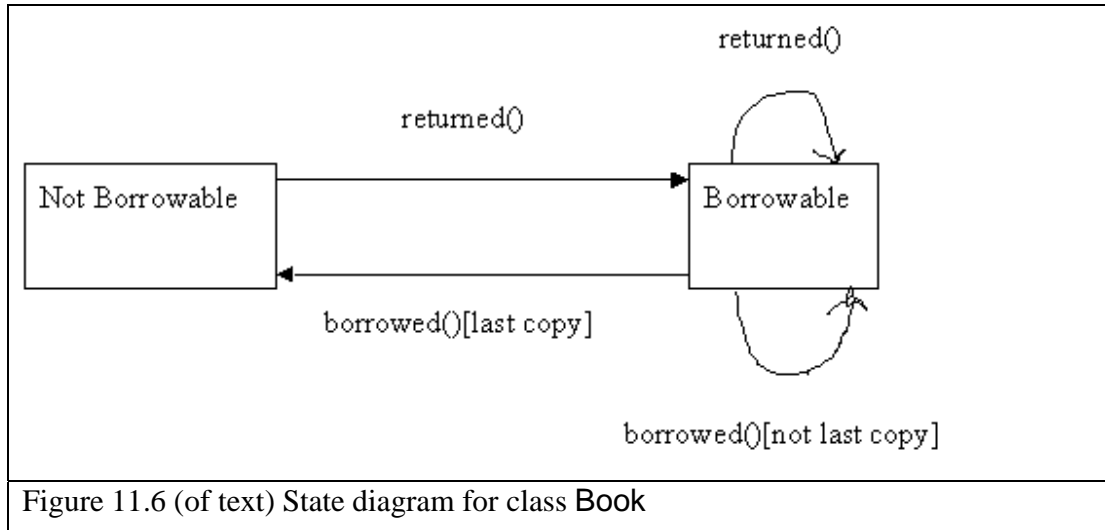


Figure 11.6 (of text) State diagram for class **Book**

-- This illustrates the situation where *sometimes an event will cause a state transition and sometimes not*. Basically, it means we have to look in more detail at the attribute values. In the above example, message `borrowed()` causes a transition to Not Borrowable only when the copy borrowed is the last copy on the shelf.

This is shown in the diagram by including two *conditions*, `[last copy]` and `[not last copy]`. The conditions guard the transition. The transition caused by receiving `returned()` is unguarded.

-- Figure 11.6 also illustrates the case of self-transitions.

On-line shopping (ex Software Design by E. Braude, Wiley, 2004)

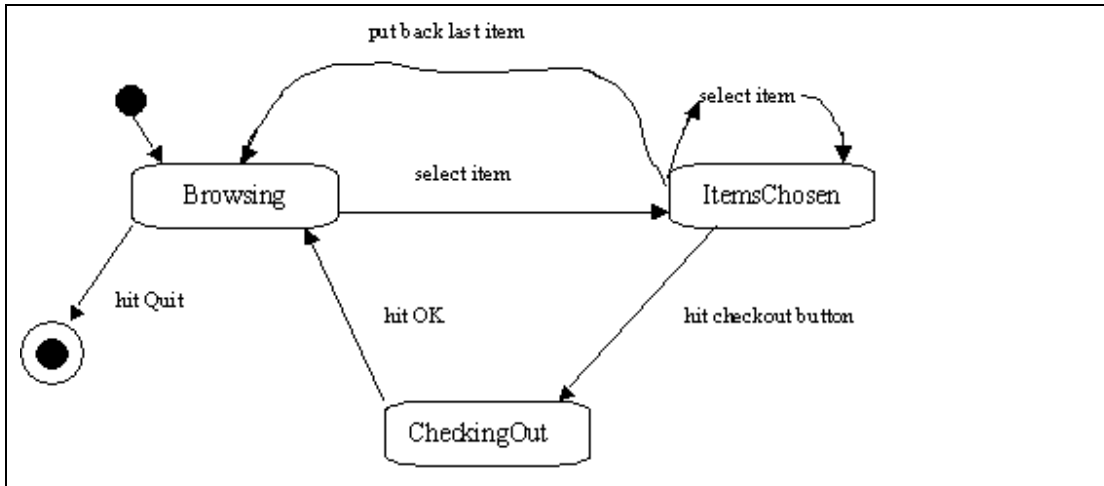


Fig A: State Diagram for OnLineShopper Class, *without* substate detail

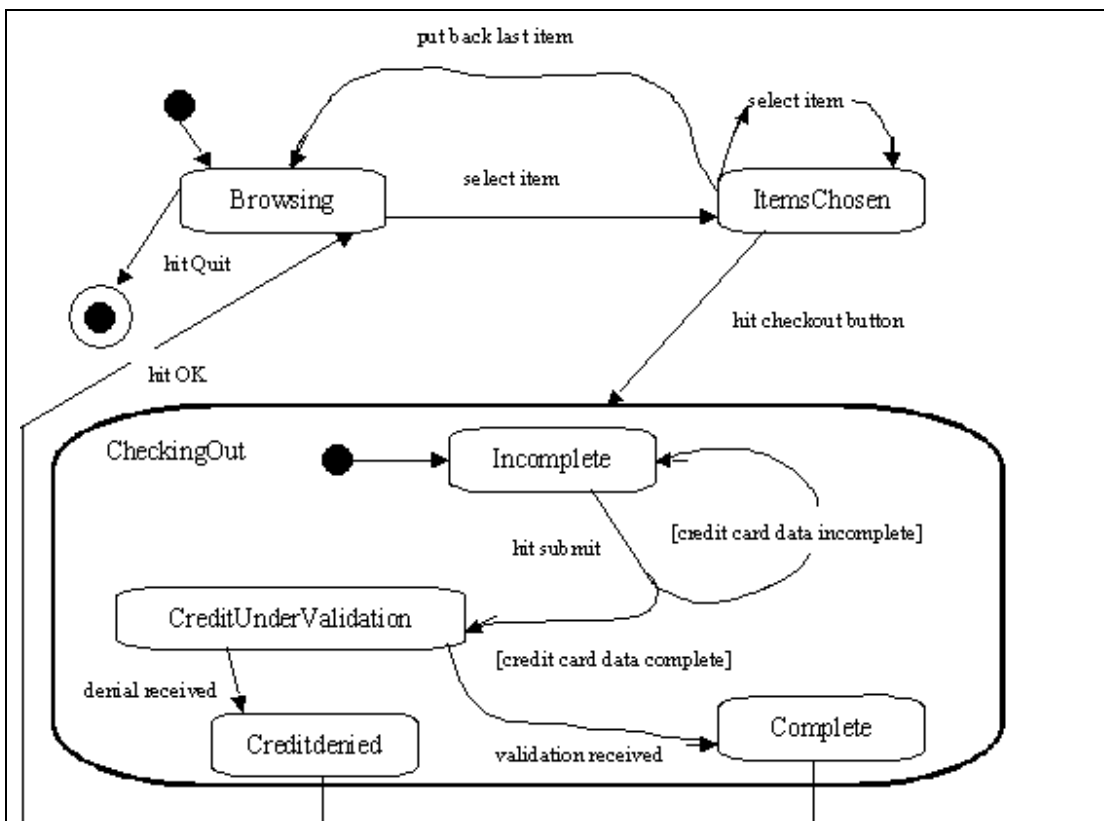


Fig B: State Diagram for OnLineShopper Class, *with* substate detail

Notes:

(1) When Shopper object enters the CheckingOut state, it automatically enters the substate Incomplete (i.e. CheckingOut.Incomplete).

(2) This author (Braude) writes about "KEY CONCEPT: State Diagrams" as follows:

"Some applications or parts thereof are conveniently thought of as being in one of several possible states. UML state diagrams help us to visualize these and the events that cause transitions among them".

Example of use of states (called modes) in an actual project

Note: The following few pages illustrate the use of states (modes) in an actual project. In that case, UML or other diagrams were not used with reliance on tables etc instead.

[START OF EXTRACT FROM AN ACTUAL PROJECT:

S-1.3-1

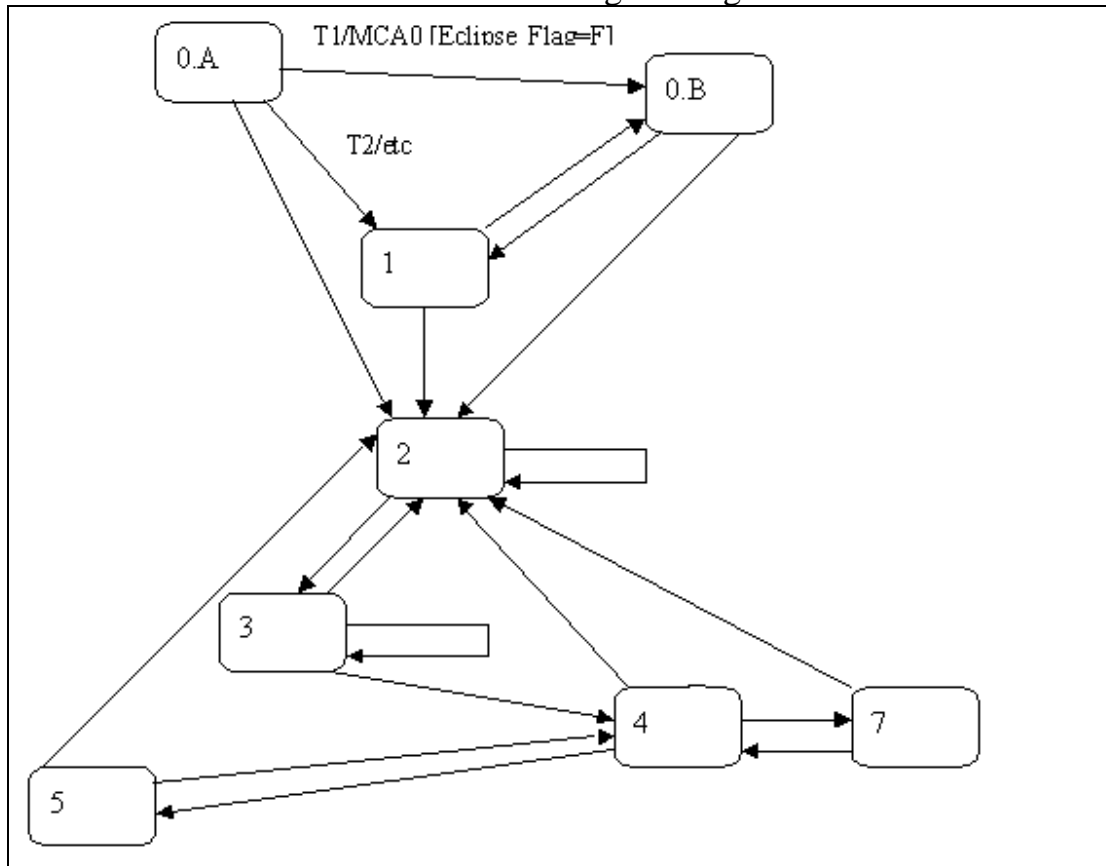
The operational modes 0 (IAM,SBM), 1(ISA), 2(SSA), 3(STA), 4(IPS), 5(TCM), and 7(OGM) shall be provided - see **mode attributes** in dictionary for details.

TRACE: F3.1.2.1-1, #

Corresponding allowable mode transitions were defined as:

mode	{{0A,0B}, {0A,1},{0A,2},	Labelled as { T1, T2, T18,
transitions	{0B,1}, {0B,2},	T3, T19,
allowed	{1,0B},{1,2},	T20, T4,
	{2,2}, {2,3},	S4, T5,
	{3,2}, {3,3}, {3,4},	T13, S5, T6,
	{4,2}, {4,5}, {4,7},	T14, T7, T9,
	{5,2}, {5,4},	T15, T10,
	{7,2}, {7,4}}	T17, T12}

Can represent above in a state diagram as follows (partially completed):



Corresponding actions and mode entry conditions were specified as follows (in abbreviated form!):

S-1.3-4

A **mode transition request** shall not be granted unless the following **mode transition conditions** are satisfied:

Transition	Condition
T1	Eclipse_Flag = F
T2	Eclipse_Flag = F
T3	Eclipse_Flag = F
T4	Eclipse_Flag = F
T5	Eclipse_Flag = F
T6	Eclipse_Flag = F; Eclipse_Imminent = F; STRStar1_Valid =T; STRMapping_Ended =T
T7	Eclipse_Flag = F; Eclipse_Imminent = F; STRStar1_Valid =T; STRMapping_Ended =T
T8	Not used
T9	Eclipse_Flag = F
T10	Eclipse_Flag = F; Eclipse_Imminent = F; STRStar1_Valid =T; STRMapping_Ended =T; m5stage =6;
T11	Not used
T12	Eclipse_Flag = F; Eclipse_Imminent = F; STRStar1_Valid =T; STRMapping_Ended =T; (At least one bit of FSS_raw [roll] = 1) or (At least one bit of FSS_raw [pitch] = 1);
T13	Eclipse_Flag = F
T14	Eclipse_Flag = F
T15	Eclipse_Flag = F
T16	Not used
T17	Eclipse_Flag = F
T18	Eclipse_Flag = F
T19	Eclipse_Flag = F
T20	Eclipse_Flag = F
S4	Eclipse_Enable = F
S5	Eclipse_Enable = F

If the **mode transition override** flag (see dictionary) is True then the associated condition shall be interpreted as allowing the transition (regardless of any derived values of the condition). If a **mode transition request** is not granted then one of **errors (recoverable)** shall be generated.

Remark: **errors (recoverable)** is assumed to include mode transitions.

TRACE: F3.1.2.1.2-4,5,6,8, #

Each mode transition shall have associated **mode transition actions** as follows:

Transition	Action
S4	MCA0; IMU_A[yaw]=0; MA3; m2SSA_Control=T; m2SSA_Tranquilisation = F; RGA18Omega_Ref=0;
S5	MCA0; IMU_A[yaw]=0; m3Start_Map_Flag=F; RGA_Flag=0; m3Accum_Flag=F
T1	MCA0;
T2	MCA0; MA1; MA2; IMU_A[yaw]=0; RCSMin_on_time=RCSDefault_Min_on_time;
T3	MCA0; MA2; IMU_A[yaw]=0; RCSMin_on_time=RCSDefault_Min_on_time;
T4	MCA0; MA4; IMU_A[yaw]=0; m2SSA_Control=T; m2SSA_Tranquilisation = F;
T5	MCA0; IMU_A[yaw]=0; m3Start_Map_Flag=F; RGA_Flag=0; m3Accum_Flag=F
T6	MCA0;MCA2; RGA_Flag=3; IMUYaw_Gyro_Delay_Counter = IMUDefault_Gyro_Delay; FCV LCL control word[w]=0,{w,1,4}; m4Slew_authorise=F; m4Slew_Ended=F; m4Slew_started=F; m4H = 0; m4Slew_Abort=F; RGA7Offset_Ended =F; RGA7Offset_Abort=F; IMUfilterUse_STR=T; IMUfilterYaw_On = 0; IMUfilterYaw_On_Rate = 0; IMUfilterPP_Filter = T;
T7	MCA0;MCA2; RGA_Flag=5; RCS off_modulation=F; RCSMin_on_time=RCSDefault_Min_on_time; RCSMin_off_time = RCSDefault_Min_off_time; MA3;m5Stage = 1; RCSTtot=0;m5Time=0; IMUYaw_Gyro_Delay_Counter = IMUDefault_Gyro_Delay; IMUfilterUse_STR=T; IMUfilterYaw_On = 0; IMUfilterYaw_On_Rate = 0; IMUfilterPP_Filter = T;
T9	MCA0; Ref_Flag=F; m7Steer_Flag=T; IMU_A[roll]=0;
T10	MCA0;MCA2; RGA_Flag=5; IMUYaw_Gyro_Delay_Counter = IMUDefault_Gyro_Delay; FCV LCL control word[w]=0,{w,1,4}; m4Slew_authorise=F; m4Slew_Ended=F; m4Slew_started=F; m4H = 0; m4Slew_Abort=F; RGA7Offset_Ended =F; RGA7Offset_Abort=F; IMUfilterUse_STR=T; IMUfilterYaw_On = 0; IMUfilterYaw_On_Rate = 0; IMUfilterPP_Filter = T;
T12	MCA0;MCA2; RGA_Flag=3; IMUYaw_Gyro_Delay_Counter = IMUDefault_Gyro_Delay; FCV LCL control word[w]=0,{w,1,4}; m4Slew_authorise=F; m4Slew_Ended=F; m4Slew_started=F; m4H = 0; m4Slew_Abort=F; RGA7Offset_Ended =F; RGA7Offset_Abort=F; IMUfilterUse_STR=T; IMUfilterYaw_On = 0; IMUfilterYaw_On_Rate = 0; IMUfilterPP_Filter = T;
T13	MCA0; IMU_A[yaw]=0; MA3; m2SSA_Control=T;

	m2SSA_Tranquilisation = F;
T14	MCA0; IMU_A[yaw]=0; MA3; m2SSA_Control=T; m2SSA_Tranquilisation = F;
T15	MCA0; IMU_A[yaw]=0; MA3; m2SSA_Control=T; m2SSA_Tranquilisation = F;
T17	MCA0; IMU_A[yaw]=0; MA3; m2SSA_Control=T; m2SSA_Tranquilisation = F;
T18	MCA0; MA1; IMU_A[yaw]=0; MA4; m2SSA_Control=T; m2SSA_Tranquilisation = F;
T19	MCA0; IMU_A[yaw]=0; MA4; m2SSA_Control=T; m2SSA_Tranquilisation = F;
T20	None

TRACE: F3.1.2.1.2-9, F3.1.2.1.2.3.2-3(last sent not TC), F3.1.2.1.2.3.2-5(last 2 lines not TC),F3.1.2.1.2.3.5-4(sent. 2), #

END OF EXTRACT FROM AN ACTUAL PROJECT]

Some other points on state diagrams:

- Here is how another text-book introduces state diagrams:

"Some applications or parts thereof are conveniently thought of as being in one of several possible states. UML state diagrams help us to visualize ..."

- State diagrams should be as simple as possible. Otherwise, with complex state diagrams, get problems:

- Harder to understand
- Harder to write code correctly for - end up with many conditional sections
- Harder to test
- Harder for external code to use the class correctly

• Activity Diagrams**General**

- Describe how activities are coordinated, whether

- to show how an operation is implemented in order to achieve a number of related things

or

- to describe carrying out of a "use Case" and how it may depend on other "Use Cases" [Example of Work Flow modeling in Business Modeling]

or

- to describe other situations where there are dependencies between activities

Note: Activity diagrams incorporate much of "flow charts"; also of Nassi-Schneiderman diagrams.

Example of a “business level” activity diagram

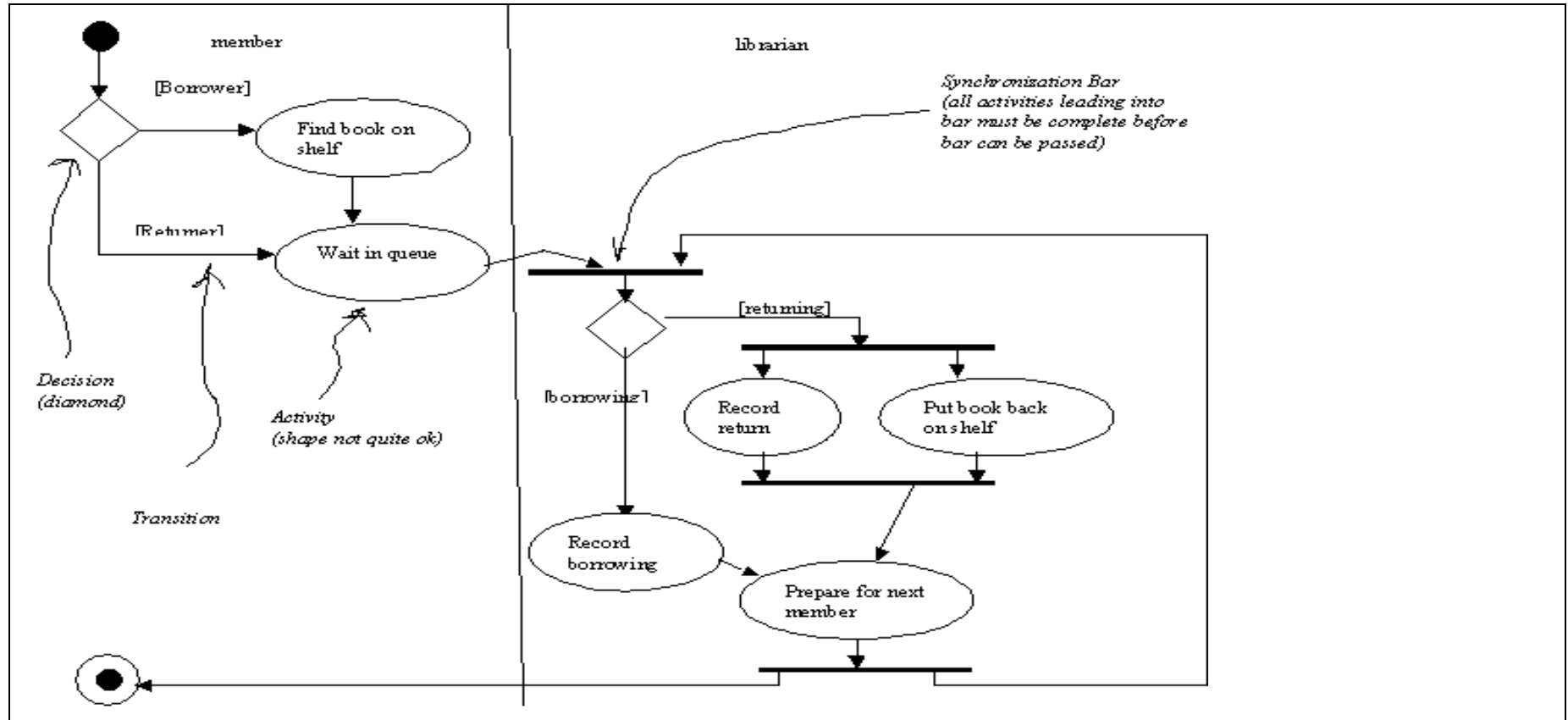


Figure 11.7 of text: Business level activity diagram of the library (describes human interaction into which system must fit)

- "Swim lanes" can be useful in arranging activity diagrams to be clearer for human readers (as above by who does what)

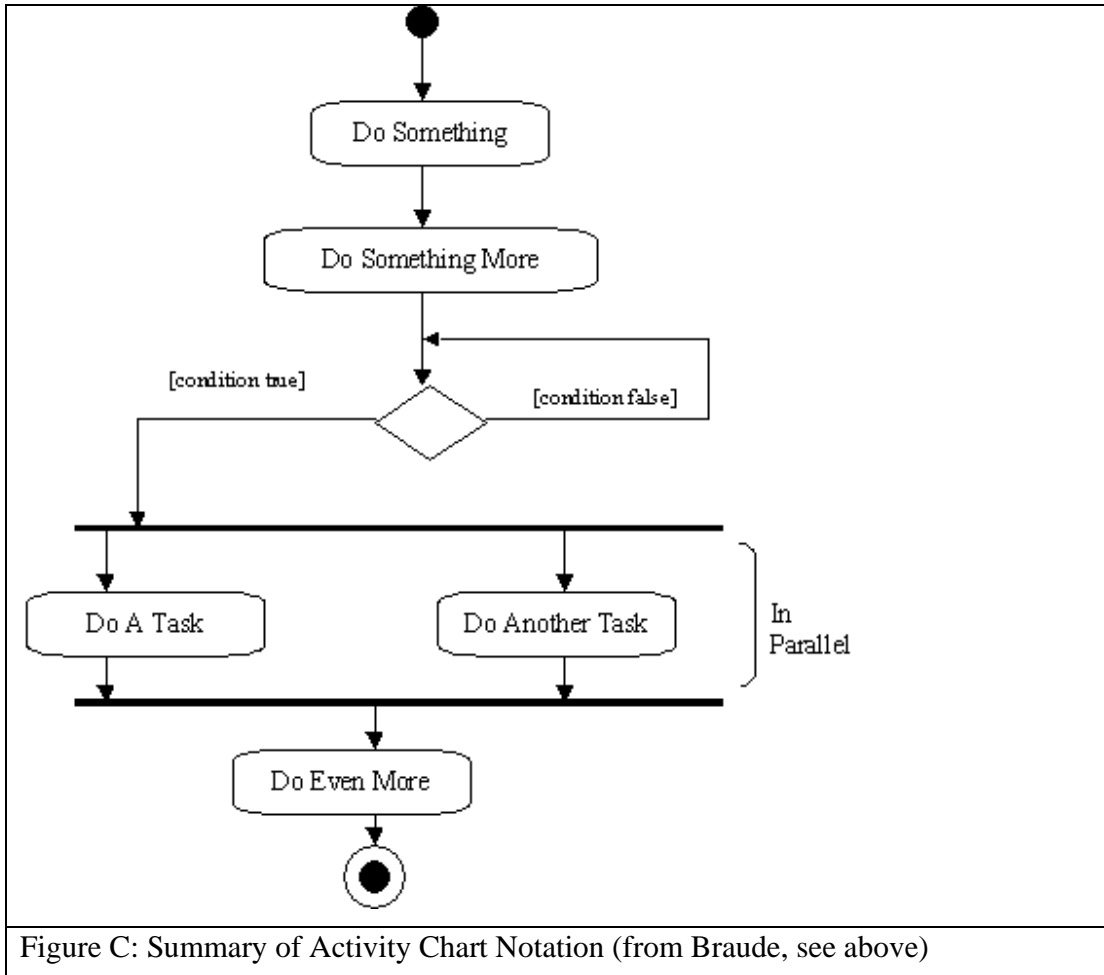
Summary of activity chart notation

Figure C: Summary of Activity Chart Notation (from Braude, see above)

Note2: Web-ref. to Nassi-schneiderman diagrams & Flow Charts

http://users.evtek.fi/~jaanah/IntroC/DBeech/3gl_nassi.htm

(This is one reference among many)

Note: Chapter 12 omitted for now.