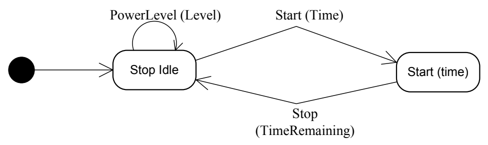
Statecharts

Mª Ángeles Martínez Ibáñez
University of Bergen

INDEX

1.- INTRODUCTION

Definition: Statecharts are used to describe the behaviour of a system and describe all of the possible states of an object as events acting upon it.



`Simple example –Microwave StateChart-`

1.- INTRODUCTION

As it's seen, Statecharts are finite state machines extended with hierarchy and parallelism, allowing a complex system to be expressed in a more compact way.

Statecharts describe how the functionality of a system relates to the state in which it exists at any one point in time, and how its state changes in response to events acting upon it. It relates things happening to an object to changes in the object.

2.- When to use Statecharts

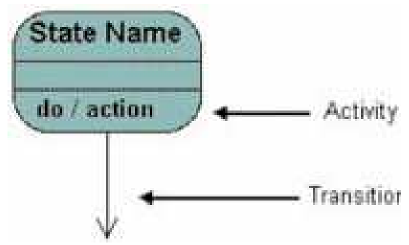
Use statecharts for classes where it is necessary to understand the behaviour of the object through the entire system.

Not all classes will require a statechart, and there aren't useful for describing the collaboration of all objects in a use case.

Statecharts are other combined with other diagrams such as interaction diagrams and activity diagrams.

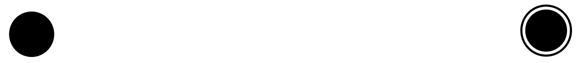
3. BASIC COMPONENTS

- **States**: they're represented by rounded boxes and show the state of the object. The arrow between states indicates the transition.



3. BASIC COMPONENTS

All the diagrams begin with a "start state" and ends with a "final state".
It's not obligatory use exactly one of each type. We can have diagrams with one or more "start state" and with no "final state". Is very common find system's behaviour which never ends.

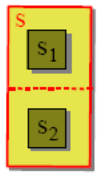


3. BASIC COMPONENTS

-> And-states: Have orthogonal components that are related by "and". They're represented graphically as rounded rectangles (parent state) divided by dashed lines

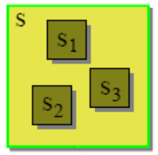
-> Or-states: This is, if the state of the high level is active, only one of the internal states will be active.

3. BASIC COMPONENTS



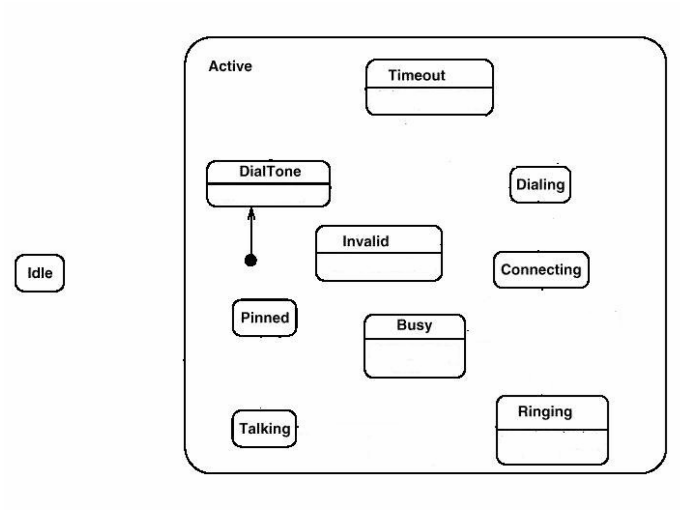
AND-state s:

s active ⇔ s_1 active and s_2 active



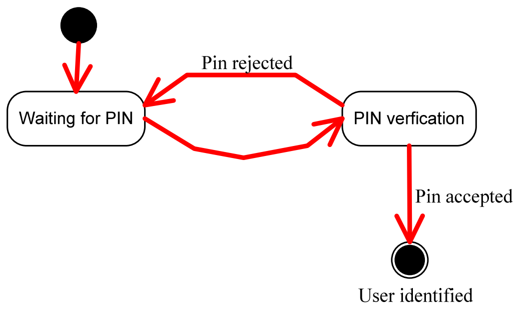
OR-state s:

s active ⇔ either s_1 active or s_2 active or s_3 active



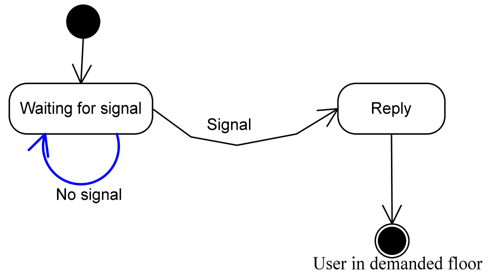
3. BASIC COMPONENTS

- **Transition:** they're represented by an arrow from one to other state.



3. BASIC COMPONENTS

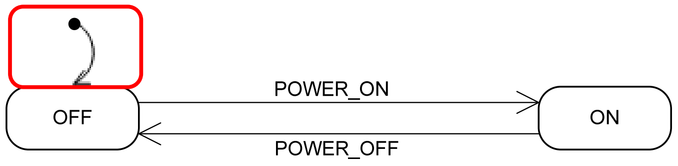
- **Self-Transition:** that is a transition which goes back to the same state.

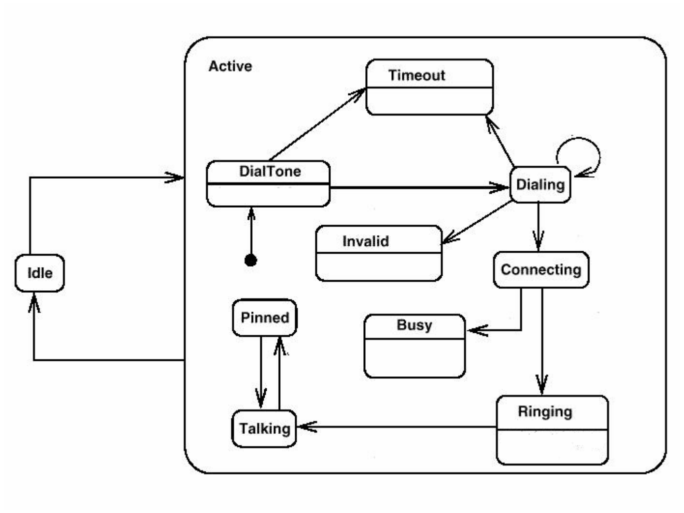


Example –Elevator-

3. BASIC COMPONENTS

❖ **Default transition:** specified by a small arrow emanating from a small solid circle.





3. BASIC COMPONENTS

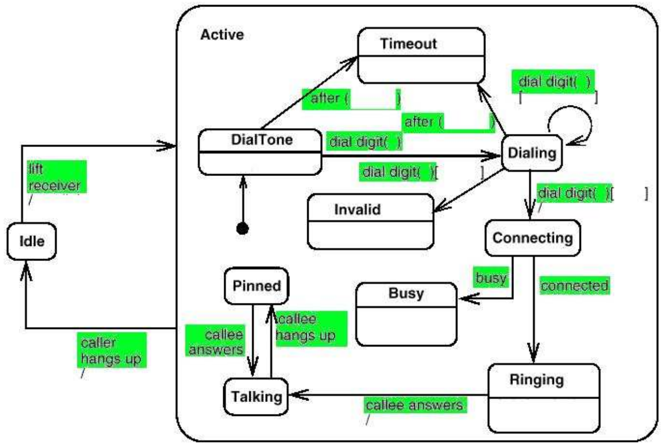
❖ **Triggers:** Triggers are the dynamic elements in statecharts. Triggers cause state transitions or static reactions. Events, conditions or a combination of both can be triggers.

Triggers represented graphically as directed graphs (arrows), connecting two states.

3. BASIC COMPONENTS

-> Events: Events signify a precise instant in time; they are edge-sensitive, comparable to signals and interrupts. Events can be generated externally to the statechart (primitive events) or internally.

Sources for internally generated events are actions, timeouts or sensors for detecting the status of states, activities, conditions and data items.



3. BASIC COMPONENTS

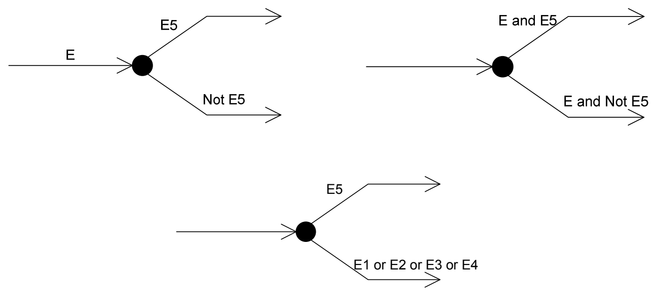
Events may also be a compound set of other events and conditions. Events are represented as a alphanumeric labels on transition lines or in the **On** field in state definitions.

3. BASIC COMPONENTS

The negation of an event using the **not** operation must be approached with caution. This negation means that the specified event did not occur.

Event expressions are evaluated according to the conventional precedence rules of logical operations, and parentheses can be used.

3. BASIC COMPONENTS



Example -Negating an event-

3. BASIC COMPONENTS

The **any** operator is used to detect the fact that some unspecified component of an event array has occurred.

The **all** operator captures the simultaneous occurrence of all events in the array.

3. BASIC COMPONENTS

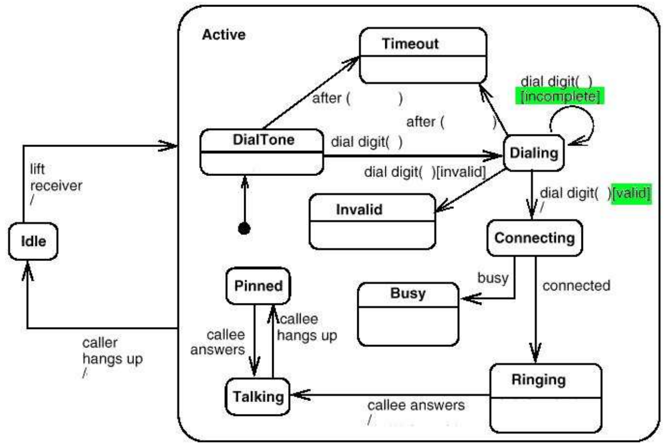
-> Conditions: Conditions are boolean expressions, valued TRUE or FALSE, that signify a time span during which the condition holds. Conditions can be edge or level-sensitive.

Conditions may be primitive elements or compound elements that express a set of boolean operations such as AND and OR.

3. BASIC COMPONENTS

Conditions are represented as alphanumeric labels on transition lines, enclosed in brackets in the form *event***[***condition***]**, or in the **Trigger** field in state definitions.

exp1 = exp2,	exp1 # exp2	`Comparison`
exp1 > exp2,	exp1 < exp2	`conditions`
exp1 <= exp2,	exp1 >= exp2	



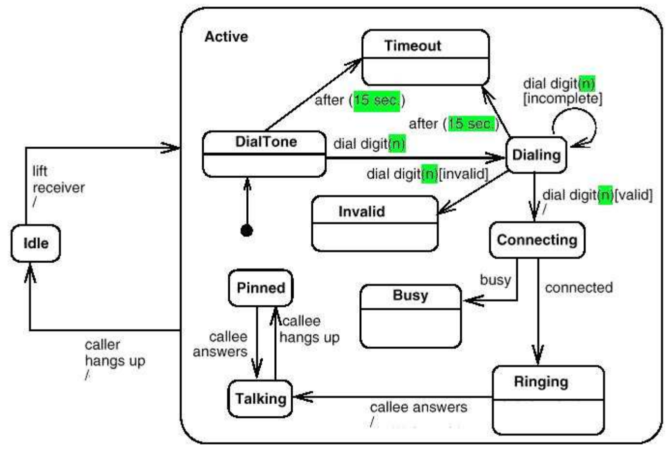
3. BASIC COMPONENTS

-> <u>Data-item expressions</u>: A data-item is a unit of information that may assume values of various types and structures. They're similar to the data elements in conventional programming languages: variables, constants, and so on. They maintain their values until they're explicitly changed and assigned new values.

3. BASIC COMPONENTS

Knowing this, data-item expressions can be used in assignment actions, and can be of different types: numeric (integer, real, bit, bit-array), strings, and structured.

With numeric expressions It's allow to use *user functions* (functions that are not predefined)



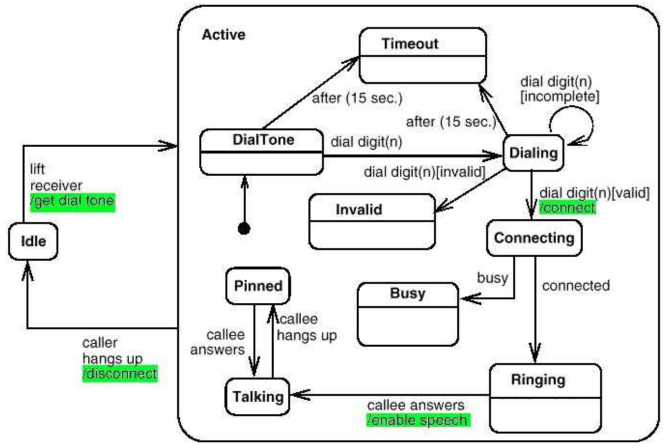
3. BASIC COMPONENTS

- **<u>Actions</u>:** Actions are instantaneous operations and are performed as a result of some trigger. Changing the value of a condition or data-item and invocations of activities are examples of actions.

  An action can also be a sequence of actions that occur simultaneously regardless of the sequence of their appearance.

3. BASIC COMPONENTS

Logical conditional actions may be expressed based on conditions or on the occurrence of events. Actions are represented as alphanumeric labels on the transition lines, separated by a slash in the form *event***[***condition***]/***action*, or in the **Action** field in state definitions. When an action is a sequence of actions, the actions that make it up are separated by semicolons.



3. BASIC COMPONENTS

If we're working with textual language, this allows various types of actions classified as follows:

- Basic actions that manipulate elements.
- Conditional and iterative actions.

3. BASIC COMPONENTS

-> Element manipulation: It causing changes that can be checked and triggering other happenings in the system.

The elements which are manipulated by most basic actions are: events, conditions and data-items.

3. BASIC COMPONENTS

Event manipulation: is just sending the event. This is perfomed by the action that is simply the name of the event.

Condition manipulation: special actions can cause a condition to become true or false.

We can find many abbreviations for this.

3. BASIC COMPONENTS

A condition has two associated events:

tr(C) changes from false to true.

fs(C) changes from true to false

An action has this events:

tr!(C) set the truth value of condition C to true

fs!(C) it corresponds with the action make_false(C), and, obviously, set it to false.

3. BASIC COMPONENTS

It’s usual to think that, tr(C) always occur when tr!(C) is executed, but this isn’t correct. The events occur only when the truth value of C changes value, but the actions can be executed without changing the truth value if ti was the desired one to start with.

3. BASIC COMPONENTS

Data-items manipulation: all types of data-items can be involved in assignments. The right-handside expression of the assignment must be type consistent with the assigned data-item on the left-hand side. Both sides must be either numeric or string.

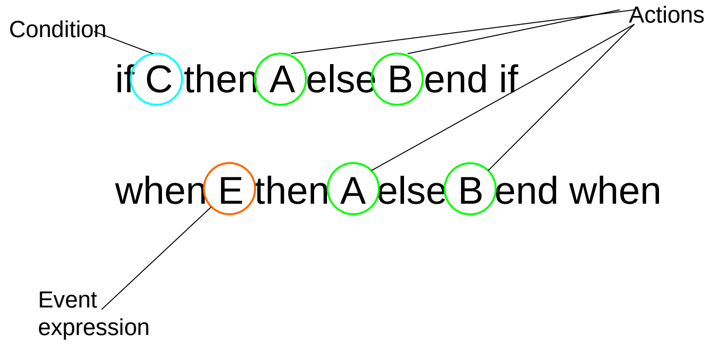
3. BASIC COMPONENTS

-> Compound actions and context variables: It's possible to perform more than one action when a transition is taken. If we want to do this, we've to write it by separating the component actions by a semicolon.

The conditional action is other kind of compound action, in thich the actual action carried out depends on a condition or an event.

3. BASIC COMPONENTS

We can find two cases:



3. BASIC COMPONENTS

-> Iterative actions: They have been declared to help manipulate arrays.

```
for I in N1 to N2 loop
        A
end loop
```

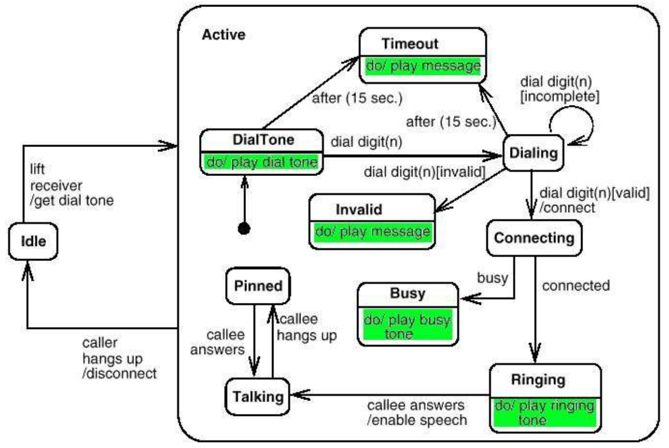
3. BASIC COMPONENTS

The action *break*, will skip the rest of loop's iterations, and the action that follows the loop construct will be the next one to execute.

The *while loop* iterates until some condition becomes false. We can use the break action here too, to jump out of the loop without completing the iteration.

3. BASIC COMPONENTS

- **<u>Activities</u>:** Activities are operations that are performed in a non-zero amount of time. Activities can be controlled (started or stopped) through actions, and their status can be monitored. Activities are not represented graphically in statecharts. They may appear as part of other element definitions.

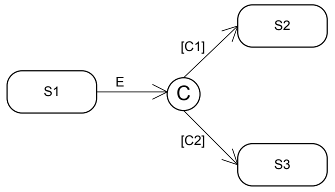


4. CONNECTORS AND COMPOUND TRANSITIONS

- **<u>Connectors and Compound Transitions</u>:** they are used to help economize in arrows to clafiry the specification.

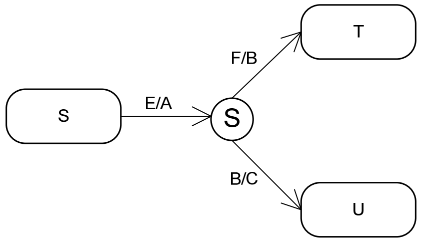
4. CONNECTORS AND COMPOUND TRANSITIONS

-> <u>Condition connectors</u>: Also called *C-connectors*. They are represented graphically as circles containing the letter C. If distinct conditions apply to the outbound transition arrows, they must be exclusive in order to prevent nondeterminism.



4. CONNECTORS AND COMPOUND TRANSITIONS

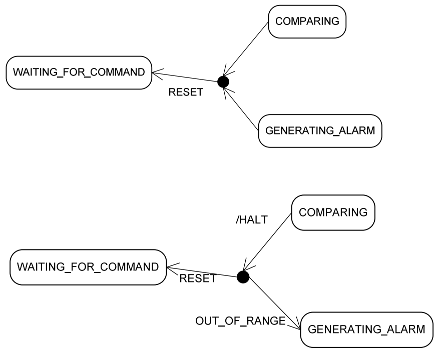
-> Swtich connectors: Also called *S-connectors*. They are usually used with events rather than conditions, but the purpose is the same one that with the C-connector.



4. CONNECTORS AND COMPOUND TRANSITIONS

-> Junction connectors: Transition arrows can be joined using this kind of connectors.

We can find events which causes exit from two states and, in the other hand, the case in which two events lead out of a state into two separate states.



4. CONNECTORS AND COMPOUND TRANSITIONS

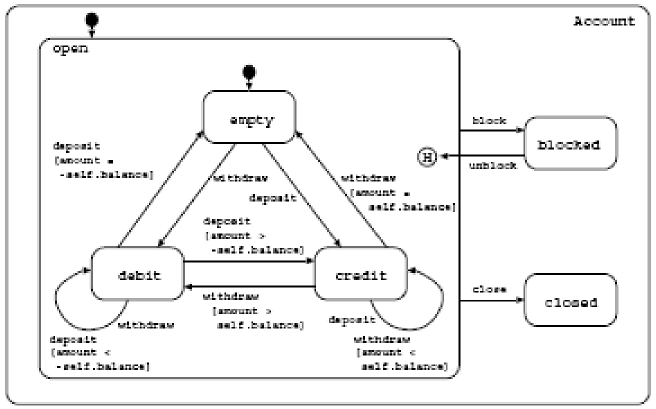
-> Diagram connectors: They are used for eliminating lengthy arrows from the chart in favor of marking two points in the chart and indicating that the arrow flows from one point to the other. Is useful for do symbolic reference to another occurrence of the same symbol in another diagram, allowing for connections among separate diagrams.

4. CONNECTORS AND COMPOUND TRANSITIONS

-> History connector: Also called *H-connector*. It's feature is to enter the state most recently visited within the group of states (this is obtained by the "history entrances").

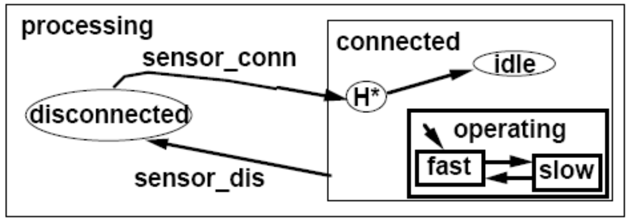
If we want to extend a history entrance down to all levels, the H-connector can appear with an asterisk attached (this is a *deep history connector*).

4. CONNECTORS AND COMPOUND TRANSITIONS


Account
open
empty
deposit [amount = -self.balance]
withdraw
deposit
withdraw [amount = self.balance]
deposit [amount > -self.balance]
debit
credit
withdraw [amount > self.balance]
withdraw
deposit [amount < -self.balance]
deposit
withdraw [amount < self.balance]
block
blocked
H
unblock
close
closed

Example –statechart with history connector for account-

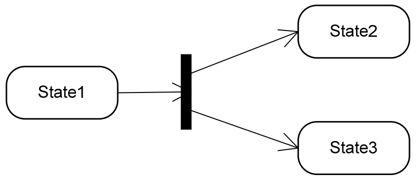
4. CONNECTORS AND COMPOUND TRANSITIONS



```
Example –if the system was last in “operating.fast”, then that
would be the state entered despite the fact “slow” is the
initial state.-
```

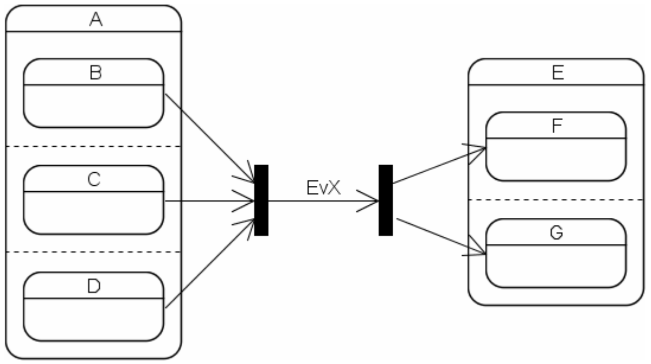
4. CONNECTORS AND COMPOUND TRANSITIONS

- **<u>Fork</u>:** We may view this as another kind of compound transition. This transition splits the execution flow into two or more parallel activities.



-Example UML Notation-

3. BASIC COMPONENTS



-Example Join&Fork-

5. CONCURRENT STATECHARTS

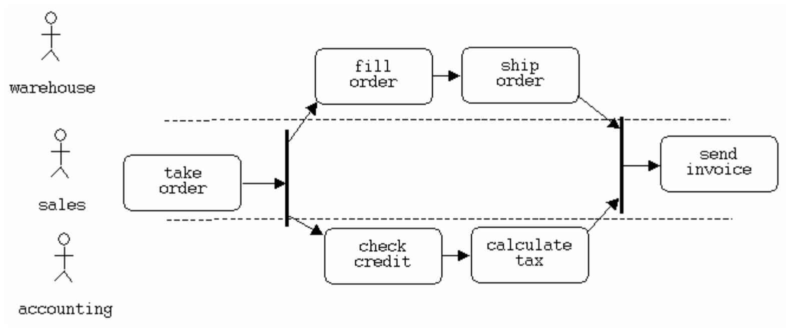
When we talk about that, statecharts support concurrent, we refer that in the same statechart, it can be more than one state executing simultaneously.

When the order leaves the concurrent states, it's in only a single state.

Concurrent state diagrams are useful when a given object has sets of independent behaviors.

But they are not very advisable.

5. CONCURRENT STATECHARTS



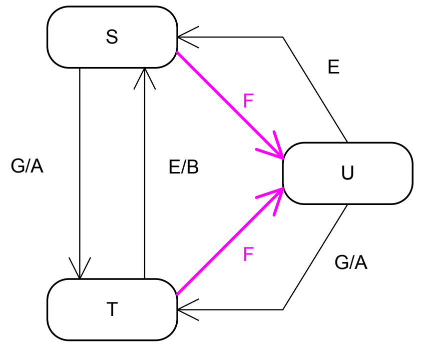
6. HIERARCHY OF STATES

The reason to use this is because the highly complex behavior cannot be easily described by simple diagrams.

Statechart can finish being a chaotic succession of states and transitions.

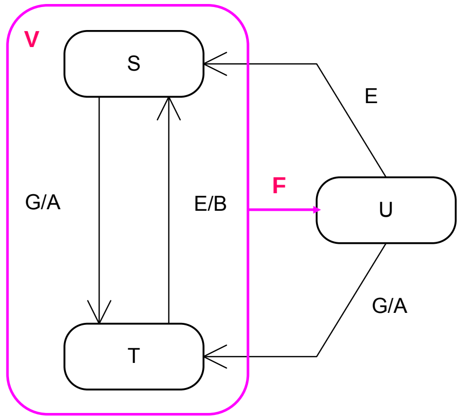
The use of a hierarchic scheme offers a solution to this problem.

6. HIERARCHY OF STATES



A simple state-transition diagram without hierarchy

6. HIERARCHY OF STATES



Clustering of states of the las example.

The semantic is to be in V is to be, exclusively, in either of its substates, S or T. This is the classical exclusive-or applied to states.

V is called an *or-state*, and it's parent of the two sibling states S and T.

SUMMARY

- **Statechart** diagrams is a graph that represents a behavior's state machine.
- A **state** is a condition during the life of an object or an interaction during which it satisfies some <u>condition</u>, performs some <u>action</u>, or waits for some <u>event</u>.

SUMMARY

- An **event** is an occurrence that may trigger a state transition.
- A **transition** is a relationship between two states indicating that an object in the first state will enter the second state and perform specific actions when a specified event occurs provided that certain specified conditions are satisfied.

SUMMARY

- The **trigger** for a transition is the occurrence of the event labeling the transition. If an event doesn't trigger any transition, it's discarded.
- A **condition** is an expression which decides whether a state transition actually occurs.
- An **action** are associated with transitions and are considered to be processes that occur quickly and aren't interruptible.

SUMMARY

- **Activities** are associated with states and can take longer. An activity may be interrupted by some event.

7.- Bibliography

[FS98] M. Flower, K. Scott: “UML Distilled. Second Edition. A Brief Guide to the Standard Object Modeling Language”, Addison Wesly Professional.

[HM98] D. Harel, M. Politi “Modeling Reactive Systems with Statecharts. The statemate approach”, McGraw-Hill, New York, United states

[Op97] http://www.opengroup.org/

[In02] http://www.informatik.uni-freiburg.de/

[De02] http://dependable.kaist.ac.kr/

7.- Bibliography

[Iv04] http://www.iv.cs.tu-berlin.de

[Cs] http://www.cs.uiowa.edu/

[Cs] http://www.cs.sjsu.edu/