An overview on UML Statecharts

Luigi Libero Lucio **STARACE**

Università degli Studi di Napoli Federico II, Naples, Italy

luigiliberolucio.starace@unina.it

https://luistar.github.io

Context and Motivations

Modelling behaviours of (reactive) systems

UML Statecharts

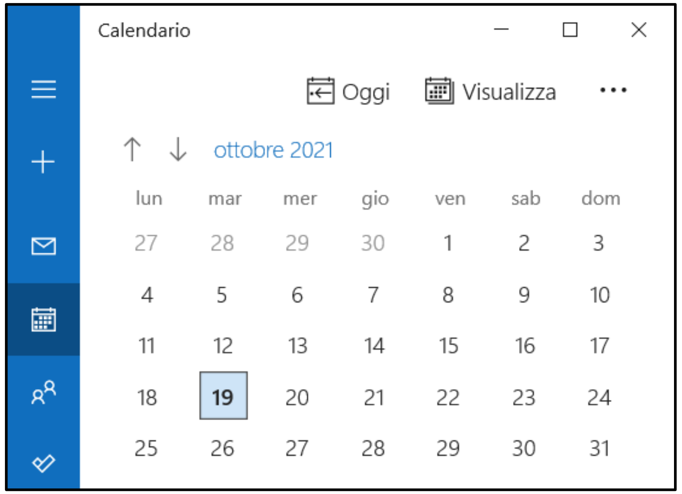
- Also known as **UML (Behavioural) State Machines**
- Extension of Harel’s Statecharts [1]
- Widely-used to model dynamic aspects of **systems** (especially **reactive** ones)

[1] Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of computer programming*, *8*(3), 231-274.

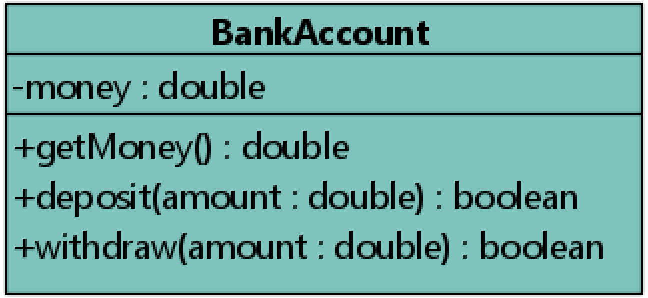
Reactive Systems

Systems that **react** to (external or internal) events

Anything comes to mind?



Software with a GUI

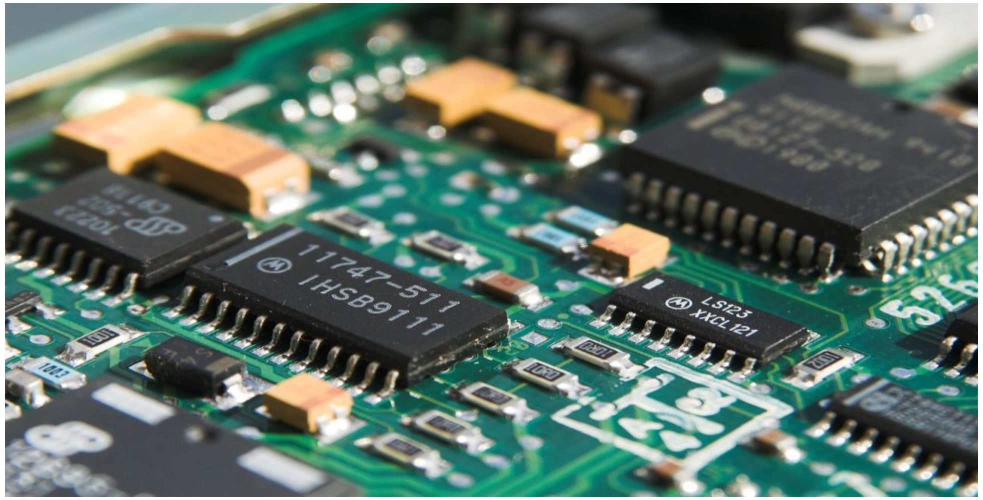


Objects in Object Oriented programming

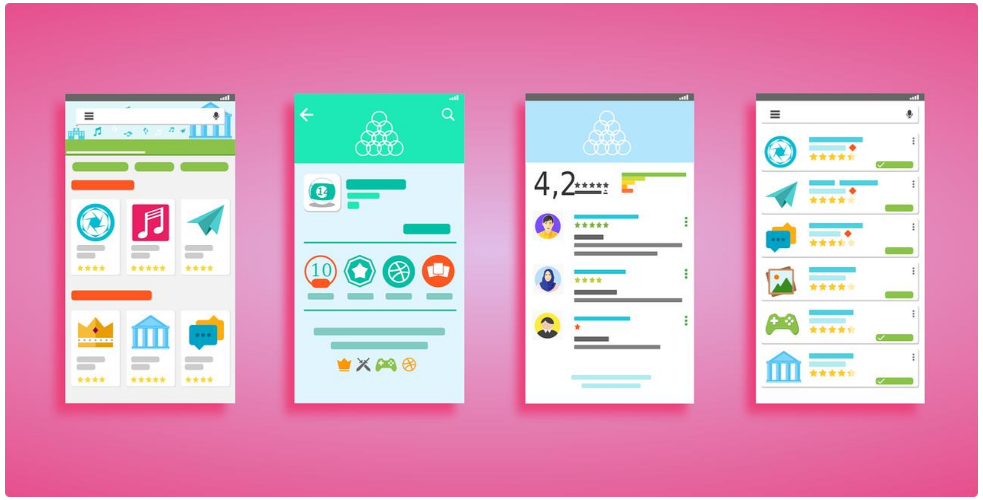
The ABS controller on the Ferrari F458

Statecharts in the real world

Statecharts are **largely** used in the industry, and not only for modelling!



Embedded software is often automatically generated from Statecharts



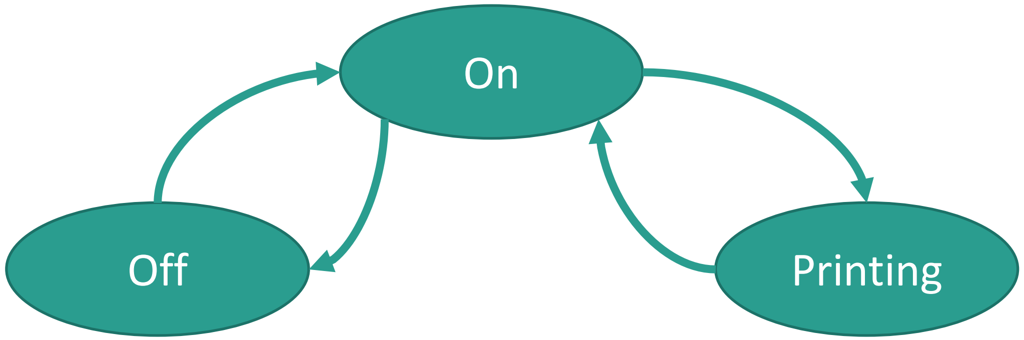
The logic behind modern User Interfaces can be managed through Statecharts

Modelling with States and Transitions

States represent situations in which some invariant condition holds.

- **Static conditions**: system is waiting for something to happen.
- **Dynamic conditions**: system is performing a specific task.

Transitions represent possible state changes



UML Statecharts

Syntax and Semantics

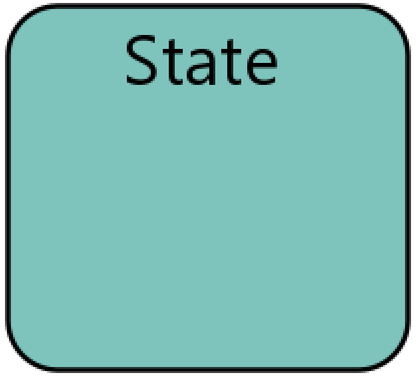
Regions, vertices and transitions

- A UML Statechart contains a top-level **region**
- A region contains **vertices** and **transitions**
- **Vertices** represent states
- **Transitions** are depicted as **directed edges** between two vertices
- Several kinds of vertices exist, with different semantics

Simple states

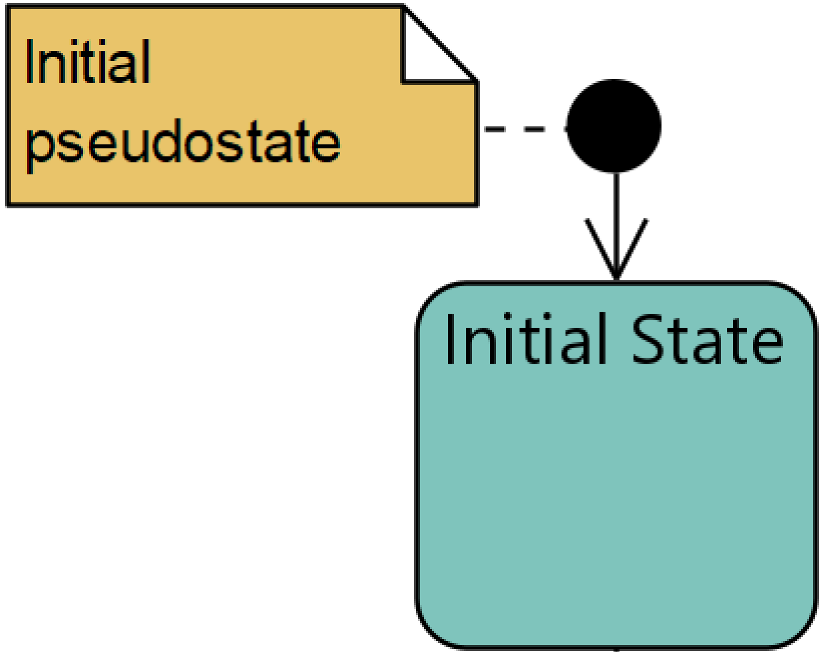
Represent unstructured system states

- Depicted as a rectangle with rounded edges
- A **name compartment** holds the (optional) name of the state, as a string.



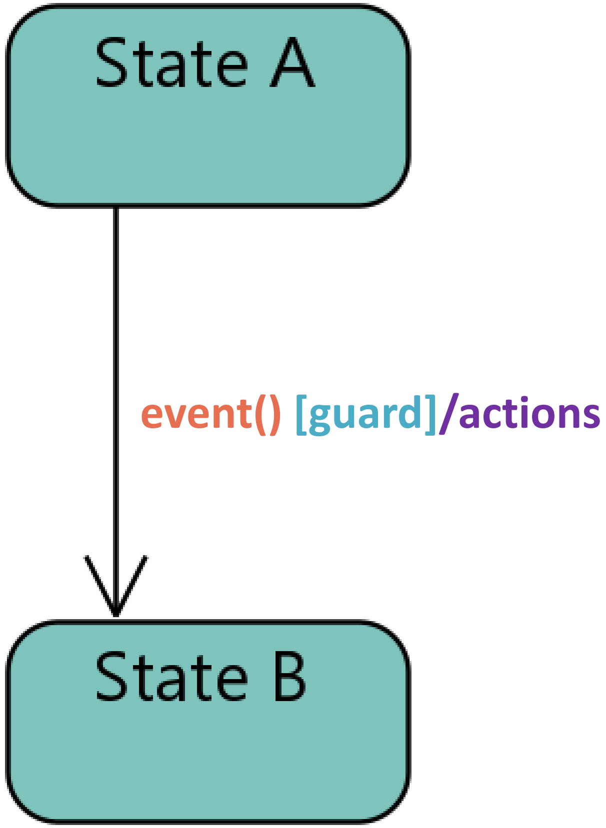
Initial Pseudostates and Final States

- **Initial pseudostates** are used to mark the default (initial) state
- A region can contain at most one initial pseudostate.
- **Final states** model a situation in which the computation is completed (i.e., the system won’t process any additional events)



Transitions: Syntax

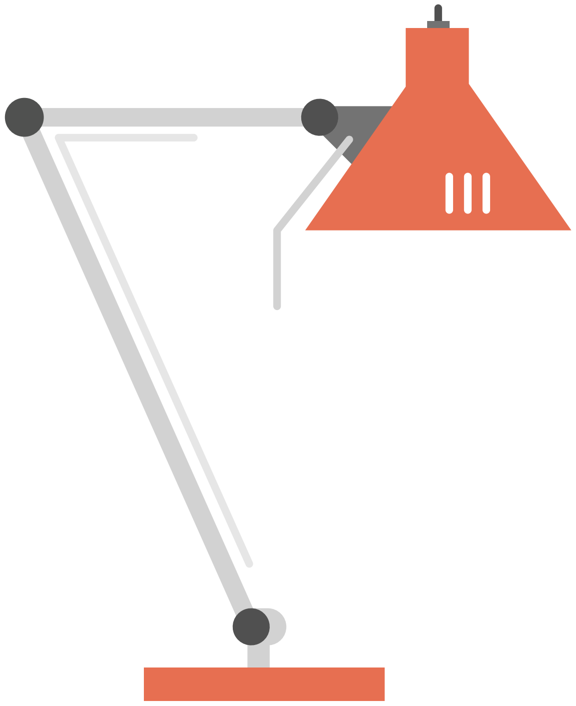
- Transitions indicate state changes
- Can be decorated with a label of the form:
  triggers [guard] /actions
  - **triggers** is a list of events that may induce a state change
  - **guard** is a Boolean condition
  - **actions** is a list of operations to execute when the transition fires.
- All the above parts of the label are optional
- Self-transitions are possible

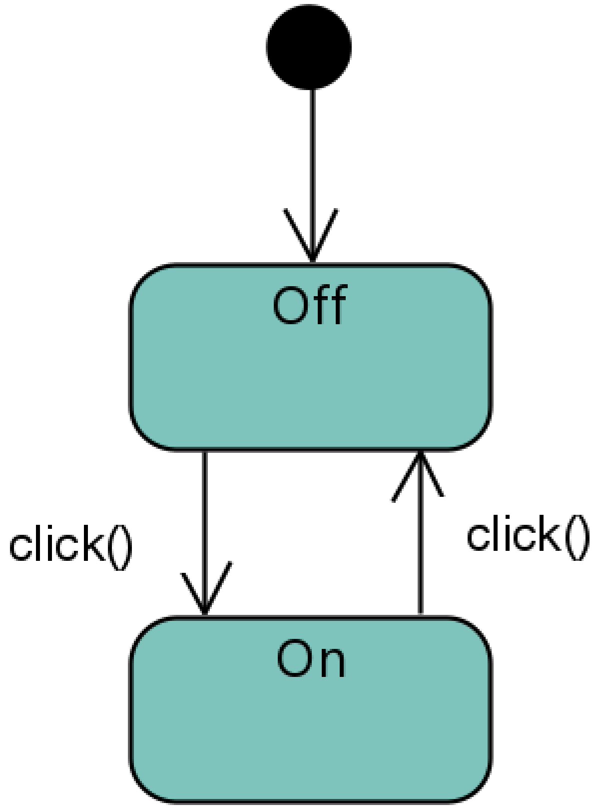


Transitions: Semantics

- For a transition to be **fireable**:
  - Events matching all of the triggers should be fired;
  - The condition in the guard must evaluate to TRUE.
- A **spontaneous** transition is one with no **triggers** and no **guard**.
- After a transition fires, its associated list of actions is executed.
- If multiple transitions are fireable, only one of them actually fires (nondeterministically determined).

Example: a lamp with a single button





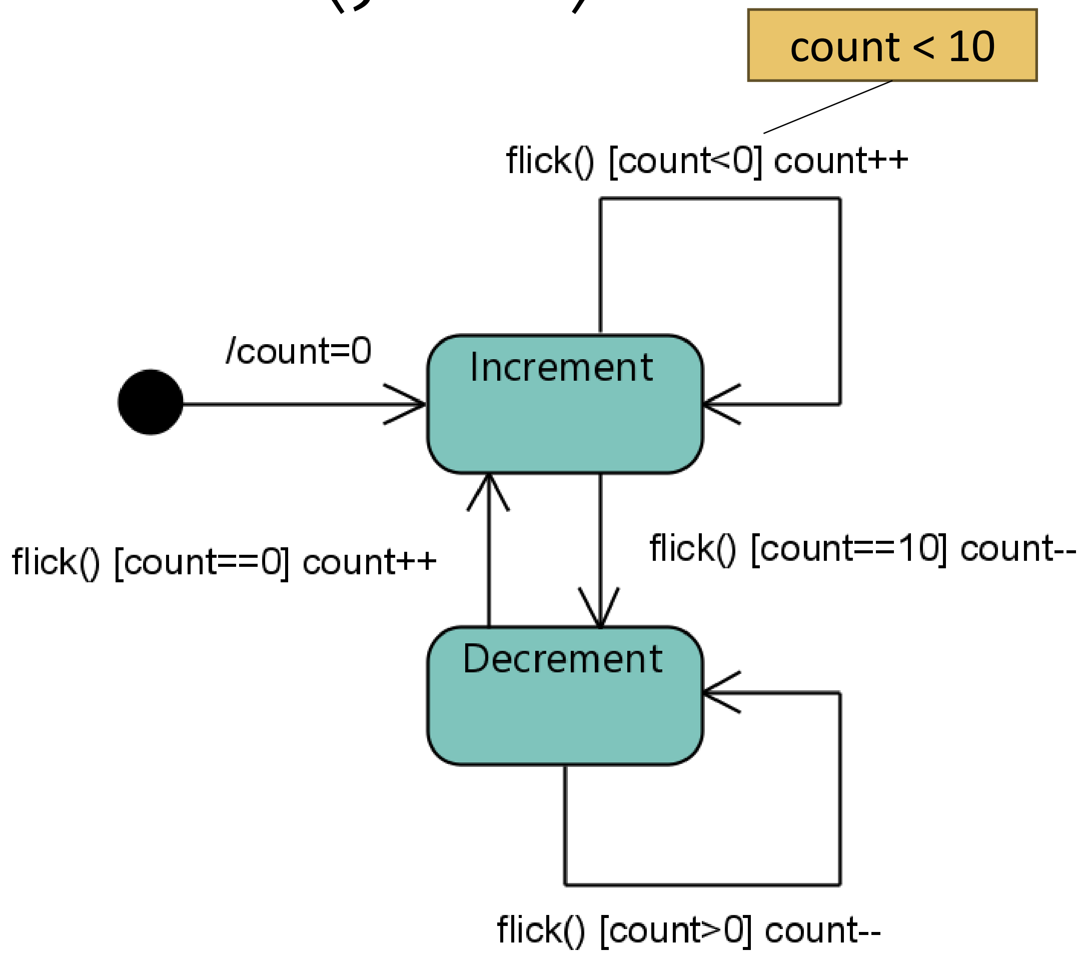
Example: a simple counter (Java)

```java
public class Counter {
  private int count = 0;
  private String mode = "increment";
  public void flick() {

    if(mode.equals("increment"))
      count++;
    else
      count--;

    if(count==10)
      mode = "decrement";
    else if (count==0)
      mode = "increment";

  }
}
```



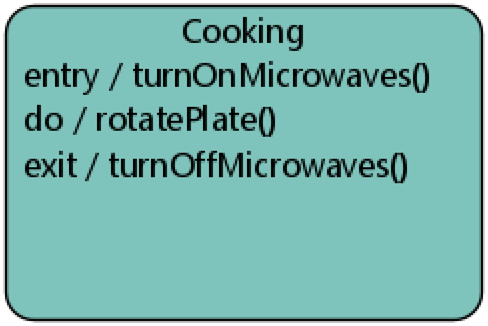
States: internal activities

States can (optionally) contain a list of **internal activities.**

Each activity is characterized by a **label** indicating **when** the activity is to be invoked.

Reserved labels:

- **entry** / activity performed upon entry
- **do** / performed as long as the system is in the state (after entry activities completed)
- **exit** / activity performed upon exit



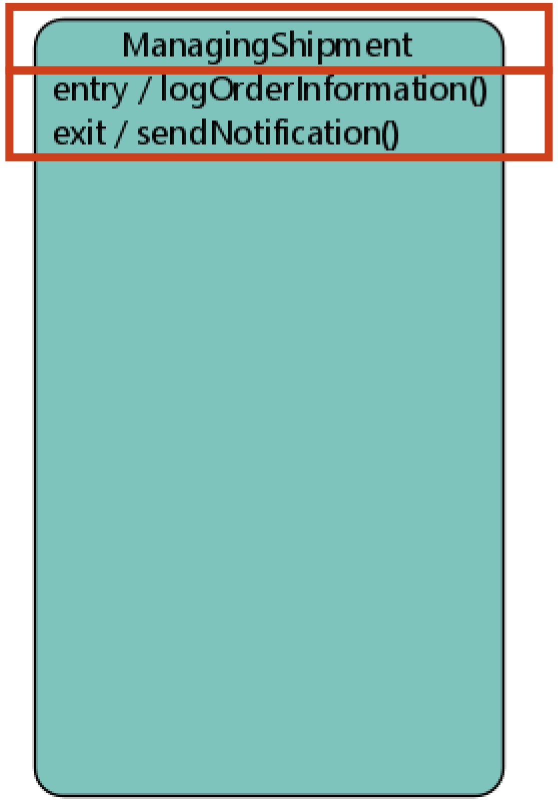
Composite States

A state can contain:

- name compartment
- internal activities compartment
- One (or more) inner regions!

A state with inner regions is a **composite state**

- States in a inner region are called **substates**



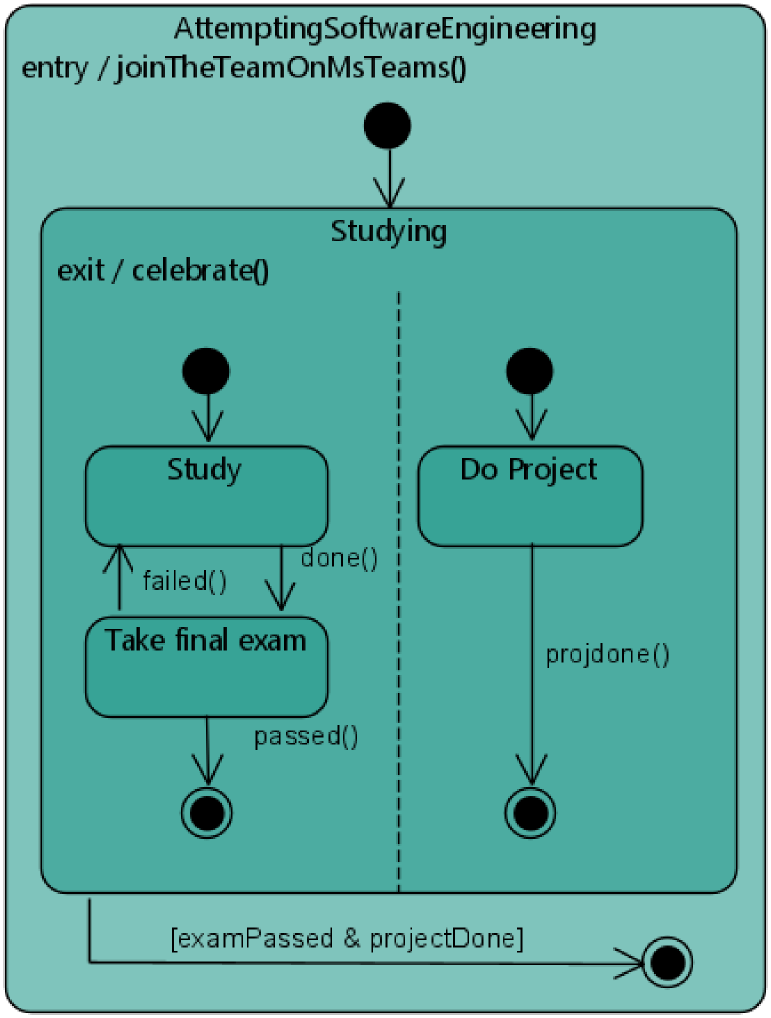
Composite States

- Allow modellers to define a **hierarchical structure**
- The inner region details the behaviour of the state it belongs to
- Provide a elegant and concise way to model complex behaviours (and hide complexity when not necessary)

The ManagingShipment composite state, with the inner region hidden

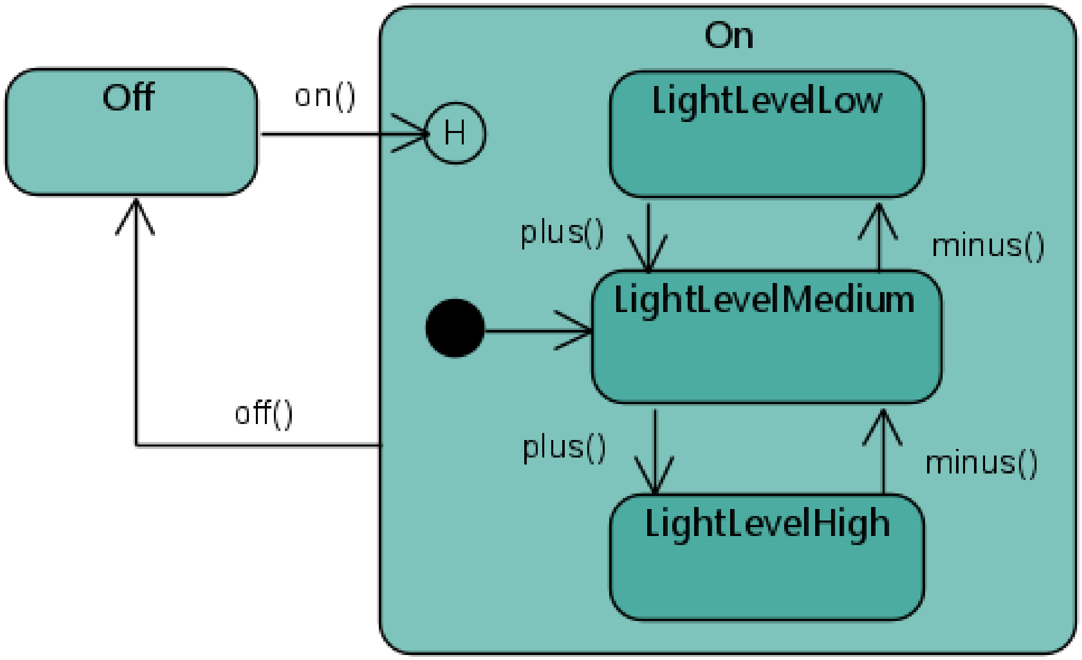
Composite States: parallel regions

- Composite states can contain multiple regions, representing behaviours that may occur in parallel
- When exiting from a composite state, all of its regions are terminated



Shallow History Pseudostates

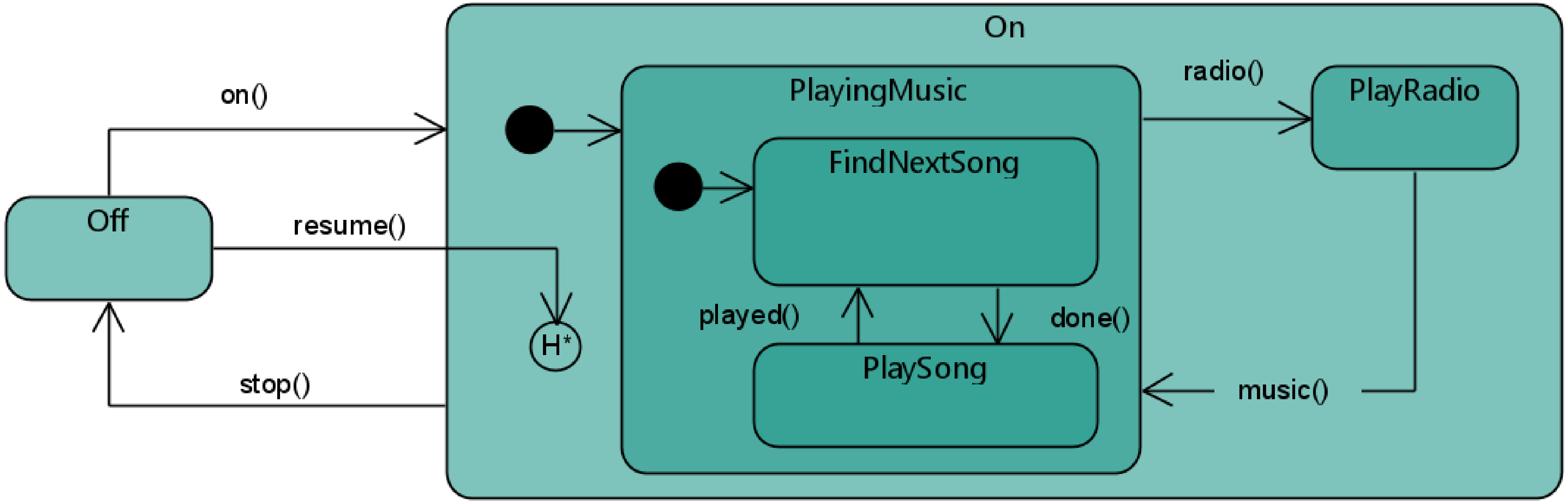
- Depicted as a H
- Represents the most recently active state of a composite state, but not substates of that substate!
- Only in composite states, and only one per region



Statechart for a lamp with three different light levels

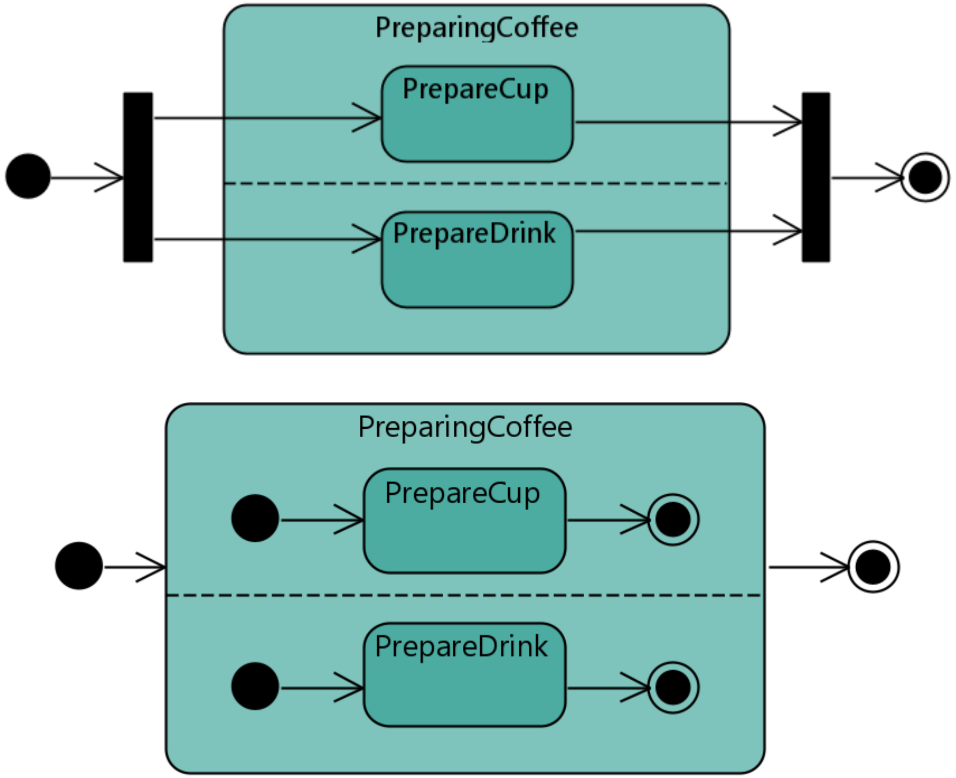
Deep History Pseudostates

- Depicted as a H*
- Same as shallow history ones, but restore the entire region configuration (substates of the substates included!)



Fork and Join Pseudostates

- **Forks** split incoming transitions into multiple transitions entering vertices in orthogonal regions
- **Joins** merge transitions exiting vertices in orthogonal regions into a single transition



Semantically equivalent statecharts.
Top one uses fork and join pseudonodes

Statecharts: Tips and Tricks

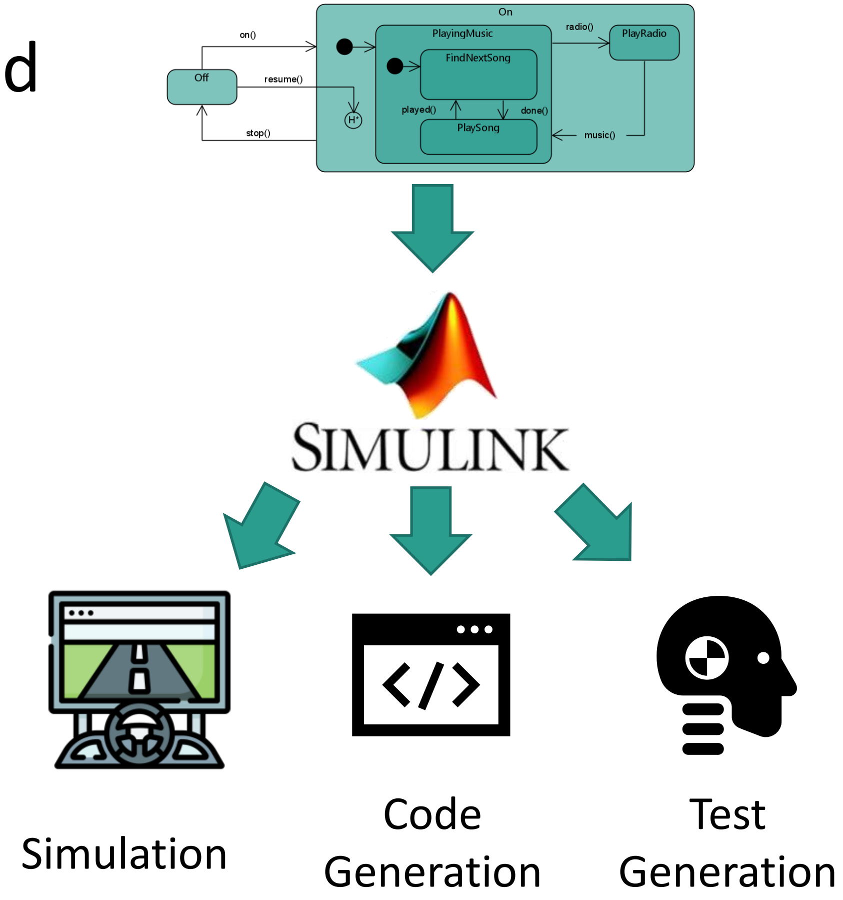
- Each state should typically have at least one transition entering it, and one exiting transition.
- Diagrams are typically read from top-left to bottom-right, so place initial/final pseudostates accordingly!
- If multiple states have a common entry and/or exit condition, consider using composite states.
- Make sure not to model non-determinism, unless it's what you *really* need!
- **At most** one state can be active in a region at any given time!

Statecharts in the wild

Practical applications of Statecharts (other than modelling!)

Model-driven Development

- Next step in the increasing abstraction trend
- De-facto standard in many embedded software domains (e.g.: automotive)
- Thanks to tools such as **Simulink**, it is possible to simulate Statechart models, to automatically generate code and tests, and much more (e.g.: formal methods!)



Model-driven Development

Pros

- In some domains, typically more cost-effective, faster and leads to higher quality
- Models understandable by domain experts
- Models are documentation!
- Less technology dependant
- Less personnel dependant

Cons

- Tools are expensive
- Not flexible enough for some applications
- Code generation typically supported for a limited number of platforms

Managing UI States with Statecharts

- Statecharts can also be used to «guide» GUI logic!
- Statecharts are easier to understand (than code!)
- Behaviour is **decoupled** from GUI components
    - Separate the **WHEN** (encoded in the Statechart) from the **WHAT** (what should happen, encoded in the UI component)
- Statecharts **scale** well as complexity grows
- Studies [2] have shown lower defect counts for statechart-based GUI controllers.

[2] *Ian Horrocks, Constructing the UI in Statecharts*

Example: Statechart-based UI with XState

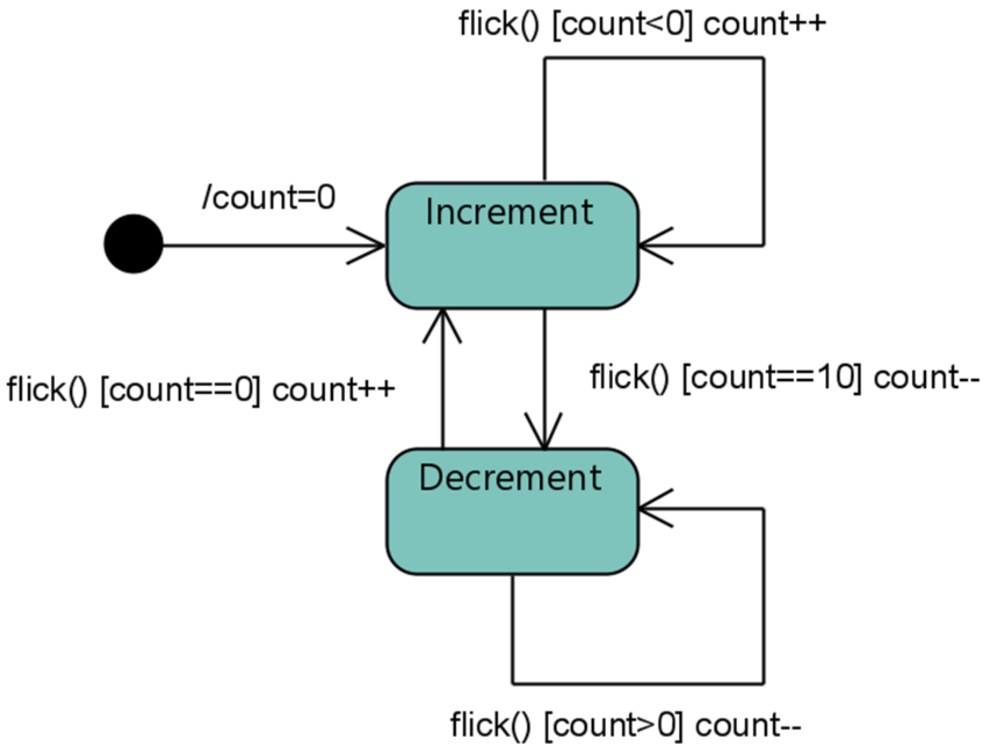
- XState is an open-source Javascript library to create, interpret, and execute statechart models
- Can be integrated with many Javascript UI libraries such as React, Vue, Svelte
- Great at managing UI State through Statecharts
- Also supports testing!
- Available at: https://xstate.js.org/

Example: Statechart-based UI with XState

Remember our Counter example?

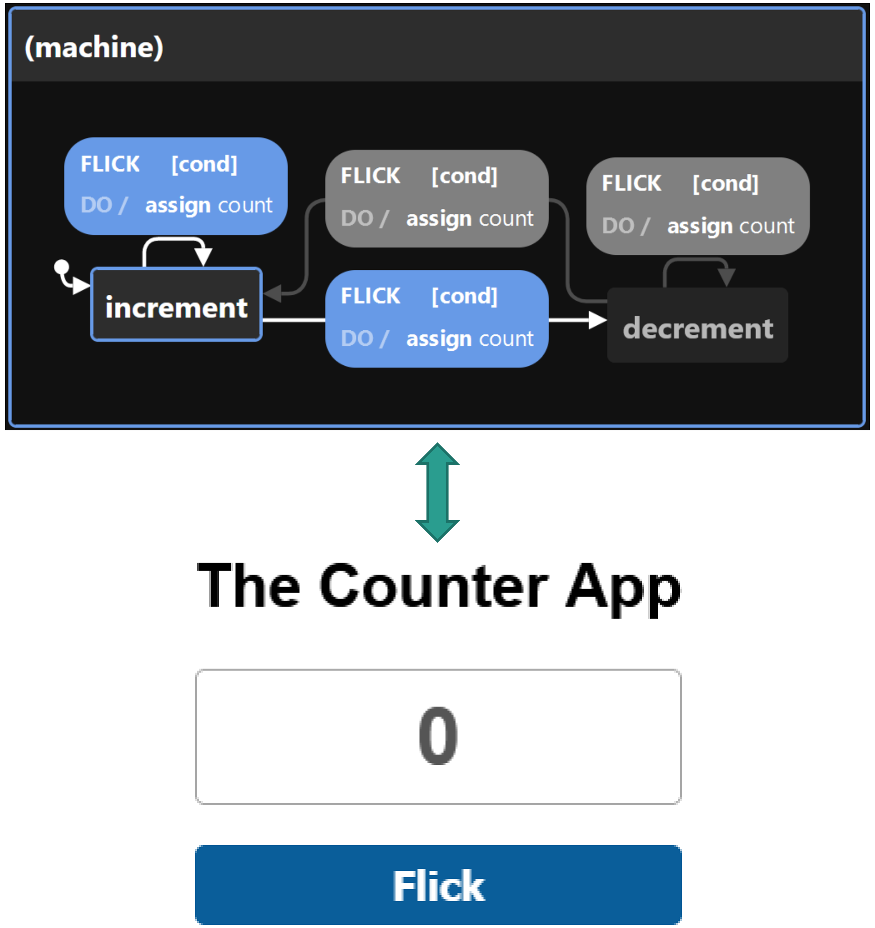
```
public class Counter {
  private int count = 0;
  private String mode = "increment";
  public void flick() {
    if(count>10)
      mode = "decrement";
    else if (count<0)
      mode = "increment";

    if(mode.equals("increment"))
      count++;
    else
      count--;
  }
}
```



Example: Statechart-based UI with XState

- Let's implement a simple UI for it, and let's do it the Statechart way, with Xstate and React
- Code Sandbox available here:



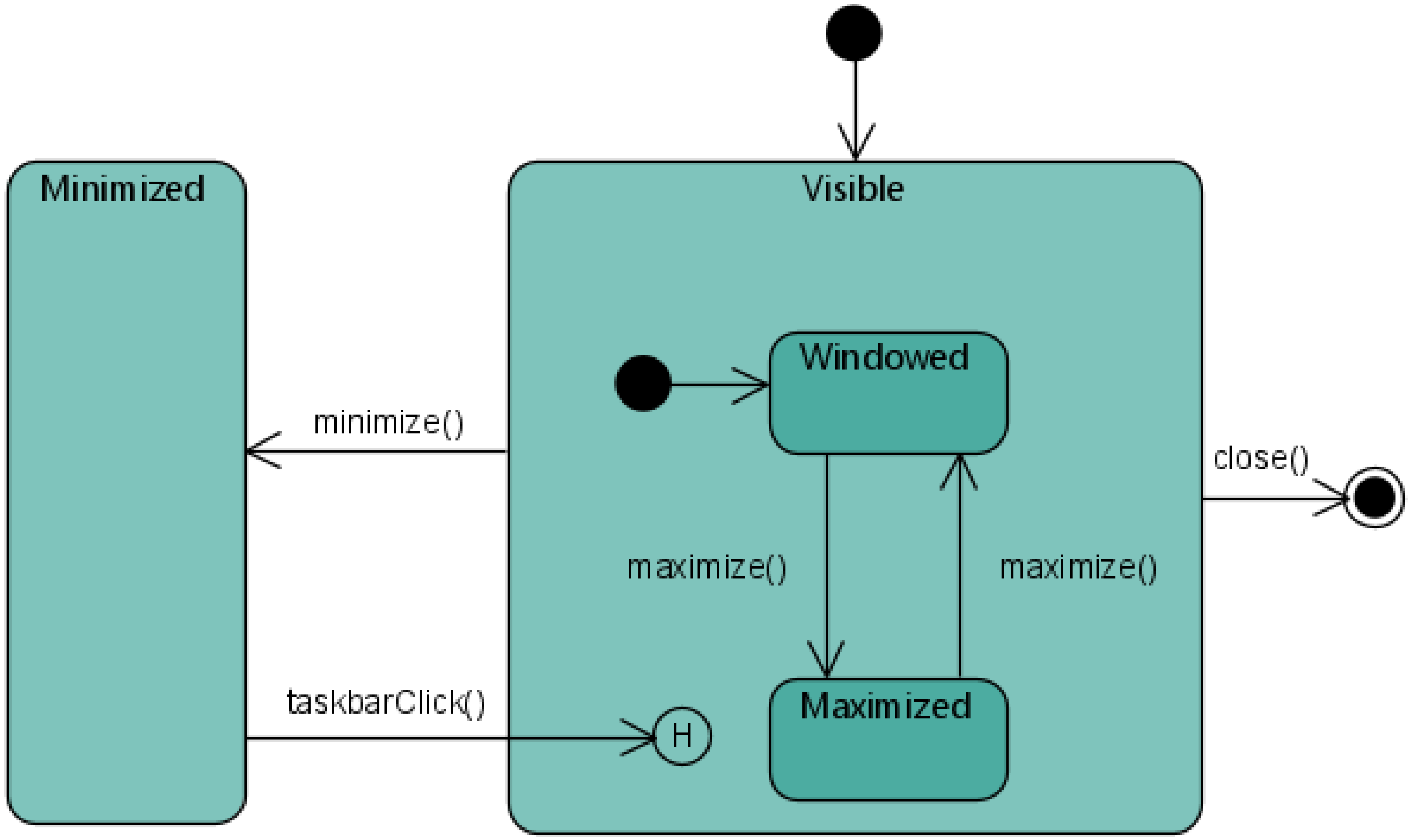
Practice time

Brave and bold volunteers, come forward!

Exercise #1

Si descriva con uno Statechart il comportamento di una generica finestra (e.g.: minimizzata, massimizzata, modalità finestra, etc.) in un ambiente desktop basato su finestre (come quello di Microsoft Windows).

Exercise #1 – Proposed solution



Exercise #2

Un orologio da tavolo, una volta acceso, mostra l’orario corrente sul proprio diplay LCD e, se l’utente preme un apposito pulsante, può anche sintonizzarsi su stazioni radio e riprodurne le trasmissioni dalle casse integrate. Tramite un pulsante «next station» è possibile passare alla stazione radio successiva, che verrà riprodotta dopo una breve fase di ricerca e sintonizzazione.

Exercise #2 – Follow up

Come si potrebbe modificare lo statechart precedente per fare in modo che, all’accensione, l’orologio da tavolo riprenda con la riproduzione della radio se la radio era attiva nel momento dello spegnimento?

Exercise #3

La schermata di login di un’applicazione permette agli utenti di inserire le proprie credenziali ed accedere. Se il nome utente inserito non è tra quelli presenti nel sistema, viene mostrato un warning dedicato. Altrimenti, se il nome è presente ma la password errata, viene mostrato un diverso warning e viene abilitato un pulsante per accedere alla funzionalità di reset password. Se le credenziali sono corrette, si accede al sistema.

Exercise #4

Una stampante, previa accensione, resta in attesa di ricevere via rete documenti da stampare. In presenza di richieste, la stampante procede alla stampa. Quando è accesa e non è in fase di stampa, la stampante, una volta al giorno, effettua la pulizia delle testine. Inoltre, sempre con cadenza giornaliera, la stampante scarica e installa aggiornamenti dalla casa madre. In questo caso, la stampante interrompe qualsiasi attività in corso per effettuare l'aggiornamento, e le riprende ad aggiornamento effettuato.

Exercise #5

Un malware, una volta installato su un PC, rimane latente fino a quando l’utente non apre Internet Explorer. A quel punto, il malware si attiva e, in parallelo, ricerca informazioni sensibili nei dischi rigidi e nella memoria del PC. Terminate queste attività, il malware sfrutta una vulnerabilità di Internet Explorer per inviare le informazioni raccolte a un server remoto. Se Internet Explorer viene chiuso prima dell’invio delle informazioni, il malware salva le informazioni trovate e riprova ad inviarle al successivo avvio di Internet Explorer.

Exercise #6 – Video Game Player

In un videogame, il giocatore inizia la partita con una salute pari a 100 HP. Durante la partita, il giocatore può subire attacchi diretti, che diminuiscono la salute residua di 5 HP. Inoltre, il giocatore può essere avvelenato. L’avvelenamento prevede una fase iniziale che dura 5 secondi, in cui il giocatore subisce 2 HP di danno al secondo, e una fase acuta, che dura 3 secondi e durante la quale il giocatore subisce 10 HP di danno al secondo. Mentre il giocatore è avvelenato in fase acuta, i danni inflitti da attacchi diretti raddoppiano. Quando la salute scende al di sotto della soglia critica di 30 HP, il giocatore passa in modalità “berserk”. Quando è in questa modalità, il giocatore è immune all’avvelenamento e si cura di 5 HP al secondo, ma subisce danni doppi dai colpi diretti. Quando i punti salute scendono a zero, il giocatore muore e la partita termina.

Exercise #7

Le luci di cortesia di un’auto hanno un interruttore che può assumere tre posizioni: ON, OFF, e DEFAULT. Quando l’interrutore è in posizione ON, le luci di cortesia sono sempre accese. Al contrario, quando è in posizione OFF, le luci di cortesia sono sempre spente. Quando l’interruttore è in posizione DEFAULT, le luci si accendono soltanto quando una delle portiere è aperta, e restano spente altrimenti. Inoltre, quando il motore è spento e l’interruttore è in posizione ON, le luci si spengono in ogni caso dopo 30 minuti per evitare di consumare la batteria, e l’interruttore si sposta su OFF.

Exercise #7 – Follow up

The proposed solution is quite complex. Is it possible to express the same behaviours with a simpler statechart?

Hint: try introducing some composite states!

References and further readings

- OMG UML Specification (2.5) https://www.omg.org/spec/UML/2.5/PDF/
- Ivar Jacobson, James Rumbaugh and Grady Booch. "The unified modeling language reference manual."