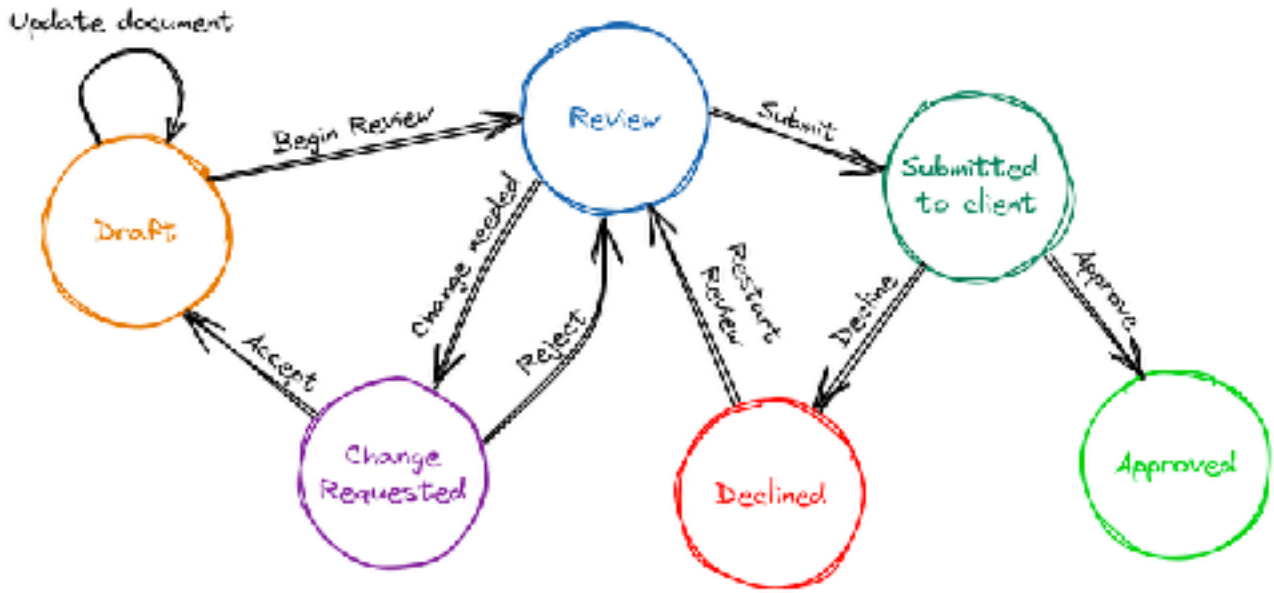
State Machine

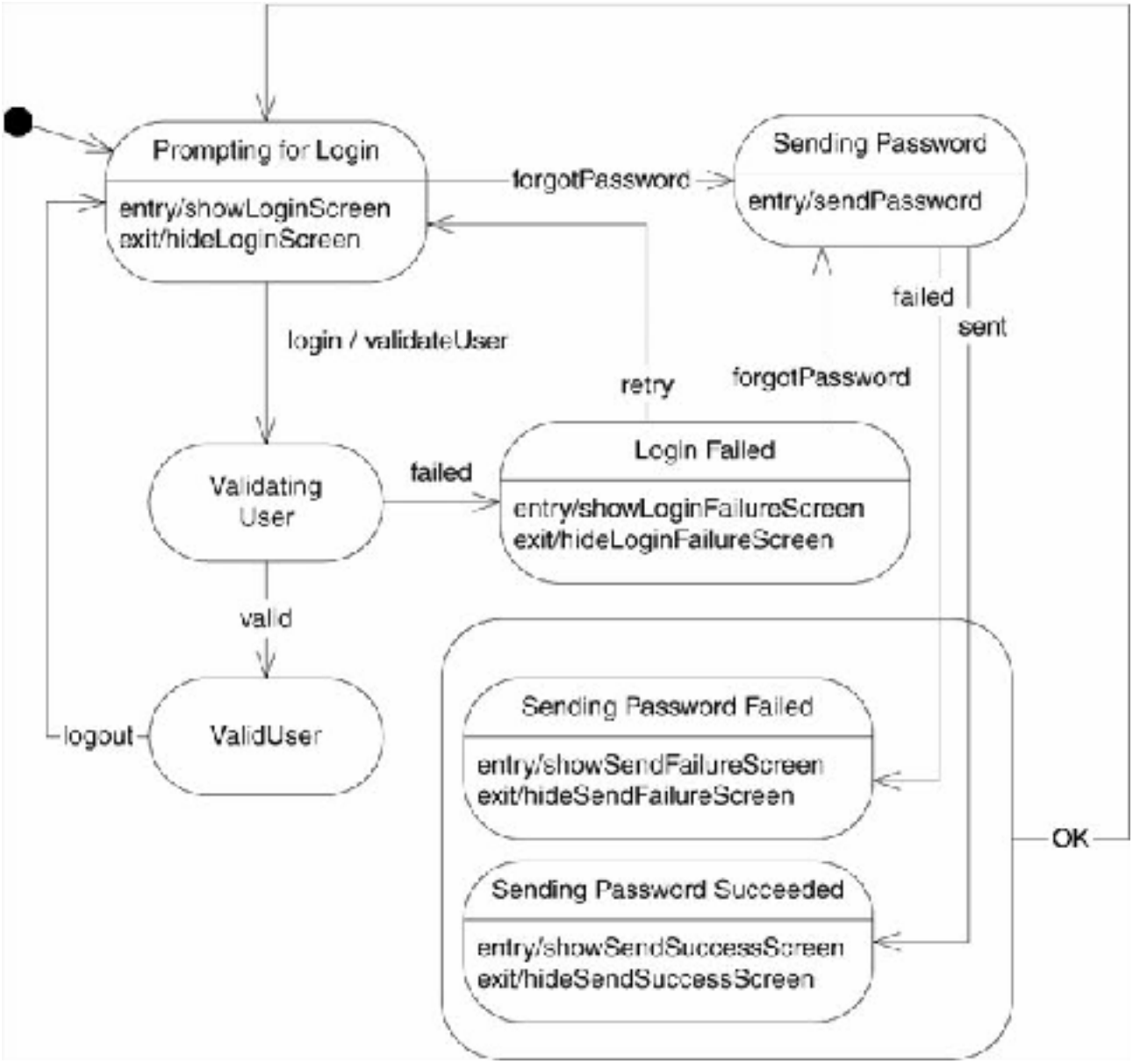
Case Study

Imagine a document which needs to be “approved” before it can be sent to a customer - to check for anything from grammar and spelling mistakes to agreed services and prices. Then, imagine that once it’s been approved it’s then sent to the customer. The customer has to approve, reject, or request changes. Whichever they choose, the document is then sent back to the approver for further review. This goes on until the document is complete and every stakeholder has approved it.



The problem

1. Before it’s either approved or sent to the customer it’s obviously in some kind of state. Call it “draft”.
2. It has to be approved or changes requested by the internal stakeholders.
3. If changes have been requested, then that branches into its own set of states.
4. The customer approves the document.
5. The customer rejects the document.
6. The legal team approves the document.



7. and so on…

The more states, the more complex the workflow. If we try model this workflow in code, we'll soon end up with dozens if not hundreds of flags to represent the "current" state.

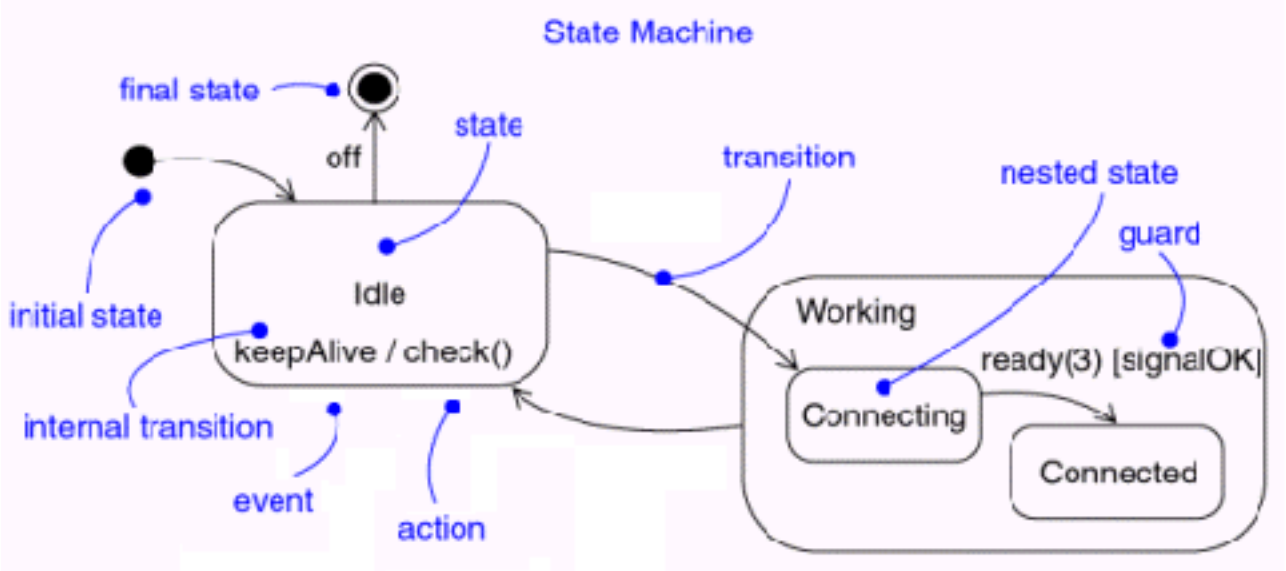
```
if (document.IsInDraft)
{
    if (!document.HasBeenReviewedByLegal)
    {
        ...
    }
    else if (document.IsSentToCustomer && document.Approved ||
document.Rejected)
    {
        ...
```

```
    }
    else
    {
        ...
    }
}
...
```

This type of code can often become *that* part of the codebase that spans hundreds or thousands of lines long and probably all in a single file too. *That* part of a codebase is often the part developers do not want to look at or work on. Developers might even try circumventing it by writing new logic in another part of the codebase thus compounding the issue even further.

Solution

An algorithm that should be reliably executed is called a Workflow. A single step or statement of a workflow is called an Activity. A standalone application or an embedded framework responsible for reliably executing workflows is called a Workflow Engine.



State machines have been around in computer science for a long time. You'll find they are especially popular in reactive systems, like

the software for video games and robotics. Designers use state machines to model a system using states, events, and transitions.

State
The basic unit that composes a state machine. A state machine can be in one state at any particular time.

Entry Action
An activity executed when entering the state

Exit Action
An activity executed when exiting the state

Transition
A directed relationship between two states that represents the complete response of a state machine to an occurrence of an event of a particular type.

Trigger
A triggering activity that causes a transition to occur.

Condition
A constraint that must evaluate to true after the trigger occurs in order for the transition to complete.

Transition Action
An activity that is executed when performing a certain transition.

Initial State
A state that represents the starting point of the state machine. A state machine workflow must have one and only one initial state.

Final State
A state that represents the completion of the state machine. A state machine workflow must have at least one final state.

A state can have an Entry and an Exit action. (A state configured as a final state may have only an entry action). When a workflow

instance enters a state, any activities in the entry action execute. When the entry action is complete, the triggers for the state's transitions are scheduled. When a transition to another state is confirmed, the activities in the exit action are executed, even if the state transitions back to the same state. After the exit action completes, the activities in the transition's action execute, and then the new state is transitioned to, and its entry actions are executed. All states must have at least one transition, except for a final state, which may not have any transitions.

Custom Activities
Building a real application will usually require creating activities that perform tasks specific to that application. Custom activities can be simple, performing just one task, or they can be composite activities containing arbitrarily complex logic. And whether they're simple or complex, custom activities can be used in lots of different ways. For example, a business application created using WF might implement application-specific logic as one or more custom activities. Alternatively, a software vendor using WF might provide a set of custom activities to make its customers' lives easier.

Persistence
When a state machine is idle, It is common to unload the state machine that have gone idle and to reload them to continue execution when a event is triggered.

An algorithm that should be reliably executed is called a Workflow. Workflow Engine ensures the reliability of workflows by persisting their state and automatically re-executing the last failed activity until the whole workflow is completed.

Workflow Engines should be able to restore any workflow state, if for example a workflow had to wait for a long time and was unloaded from memory or if a Workflow Engine node fails.

Tracking
Workflow tracking provides insight into the execution of a workflow instance. The tracking events are emitted from a workflow at the workflow instance level and when activities within the workflow execute.

Tracking enables you to monitor currently running and completed workflows. This is a very powerful asset for workflow consumers. Imagine being able to know where all your workflows are in their progress at any given time. Further, imagine being able to say something like “Give me all ‘create order’ workflows that were created in the past 12 hours and are related to customer XYZ” or “Show me all workflows that are currently stuck in the ‘waiting for confirmation’ state.”

Versioning

you want to deploy multiple version of the same workflow side by side and use the original version for each instance. For example if your business contract conditions change it makes sense to keep the older workflow instances with the original workflow definition and start new new workflows with the new definition.

Authoring

Allow declarative authoring of workflows. Declarative definition is when XML/Json is used to define the workflow structure, logic, and data and the workflow compiler turns it into physical objects. This is significant because it will allow for non-developers to be workflow authors.

Rule Engine

provide a flexible rules evaluation engine that you can use when developing Workflow. A rule is simply a declarative statement about your data. Within a rule, you declare a condition that you wish to evaluate at runtime. If the condition evaluates to true, one or more actions that you define are executed. A rule also permits you to define actions to execute when the condition evaluates to false.

Scheduling

Author should be able to start a workflow based on the following triggers.

1. When a file is dropped in a folder
2. When a email is received
3. At a particular time of the day

Dynamic Update

Dynamic update describes the ability to modify a running workflow's execution path. This opens up a new world of scenarios related to capturing missed requirements during development or allowing for exception cases where a process might need to behave differently than it normally would.

For example, consider an approval process that requires an additional round of approvals from analysts.