STATE TRANSITION DIAGRAM TUTORIAL pdf

1: State Transition Diagrams

A state diagram, sometimes known as a state machine diagram, is a type of behavioral diagram in the Unified Modeling Language (UML) that shows transitions between various objects. Using our collaborative UML diagram software, build your own state machine diagram with a free Lucidchart account today.

UML 2 Tutorial - State Machine Diagram State Machine Diagrams A state machine diagram models the behaviour of a single object, specifying the sequence of events that an object goes through during its lifetime in response to events. As an example, the following state machine diagram shows the states that a door goes through during its lifetime. The door can be in one of three states: It can respond to the events Open, Close, Lock and Unlock. Notice that not all events are valid in all states; for example, if a door is opened, you cannot lock it until you close it. Also notice that a state transition can have a guard condition attached: The syntax and conventions used in state machine diagrams will be discussed in full in the following sections. States A state is denoted by a round-cornered rectangle with the name of the state written inside it. Initial and Final States The initial state is denoted by a filled black circle and may be labeled with a name. The final state is denoted by a circle with a dot inside and may also be labeled with a name. Transitions Transitions from one state to the next are denoted by lines with arrowheads. A transition may have a trigger, a guard and an effect, as below. State Actions In the transition example above, an effect was associated with the transition. If the target state had many transitions arriving at it, and each transition had the same effect associated with it, it would be better to associate the effect with the target state rather than the transitions. This can be done by defining an entry action for the state. The diagram below shows a state with an entry action and an exit action. It is also possible to define actions that occur on events, or actions that always occur. It is possible to define any number of actions of each type. Self-Transitions A state can have a transition that returns to itself, as in the following diagram. This is most useful when an effect is associated with the transition. Compound States A state machine diagram may include sub-machine diagrams, as in the example below. The alternative way to show the same information is as follows. The notation in the above version indicates that the details of the Check PIN sub-machine are shown in a separate diagram. The following diagram shows the state machine one level up. Exit Point In a similar manner to entry points, it is possible to have named alternative exit points. The following diagram gives an example where the state executed after the main processing state depends on which route is used to transition out of the state. Choice Pseudo-State A choice pseudo-state is shown as a diamond with one transition arriving and two or more transitions leaving. The following diagram shows that whichever state is arrived at, after the choice pseudo-state, is dependent on the message format selected during execution of the previous state. Junction Pseudo-State Junction pseudo-states are used to chain together multiple transitions. A single junction can have one or more incoming, and one or more outgoing, transitions; a guard can be applied to each transition. A junction which splits an incoming transition into multiple outgoing transitions realizes a static conditional branch, as opposed to a choice pseudo-state which realizes a dynamic conditional branch. Terminate Pseudo-State Entering a terminate pseudo-state indicates that the lifeline of the state machine has ended. A terminate pseudo-state is notated as a cross. History States A history state is used to remember the previous state of a state machine when it was interrupted. The following diagram illustrates the use of history states. The example is a state machine belonging to a washing machine. In this state machine, when a washing machine is running, it will progress from "Washing" through "Rinsing" to "Spinning". If there is a power cut, the washing machine will stop running and will go to the "Power Off" state. Then when the power is restored, the Running state is entered at the "History State" symbol meaning that it should resume where it last left-off. Concurrent Regions A state may be divided into regions containing sub-states that exist and execute concurrently. The example below shows that within the state "Applying Brakes", the front and rear brakes will be operating simultaneously and independently. Notice the use of fork and join pseudo-states, rather than choice and merge pseudo-states. These symbols are used to synchronize the concurrent threads.

STATE TRANSITION DIAGRAM TUTORIAL pdf

2: Examples of State Transition Diagrams

Example of State Transition Diagram: Case Study: You need to develop a web-based application in such a way that user can search other users, and after getting search complete, the user can send the friend request to other users.

State Machine Diagram Examples State Machine Diagram Tutorial A state machine diagram is used to model the dynamic behavior of individual class objects, use cases, and entire systems. In other words, when a state machine created where the object it is attached to, that object becomes the owner of the state machine, for example, the object to be attached by the state machine could be a class, use case or even the entire system. A state machine diagram is a behavior that specifies the sequences of states an object goes through during its lifetime in response to events. A state machine are used to specify the behavior of objects that must respond to asynchronous stimulus or whose current behavior depends on their past. A state machines are used to model the behavior of objects, use cases, or even entire systems, especially reactive systems, which must respond to signals from actors outside the system. In UML, state machines introduce the two new concepts in additional to traditional start chart notation: Composite and nested states Orthogonal regions Graphically, a state is rendered as a rectangle with rounded corners. A transition is rendered as a solid directed line. Finding an online State Machine Diagram tool? Just click the Draw button below to create your State Machine Diagram online. You can also go through this State Machine Diagram tutorial to learn about State Machine Diagram before you get started. State Machine Diagram Notations State A state is a condition during the life of an object which it may either satisfy some condition for performing some activities, or waiting for some events to be received. A state has five parts: State Name â€” Name of State Entry â€” Action performed on entry to state Do Activity â€” Action performed on entry to state Exit State â€” Action performed on leaving state Deferrable Trigger â€” A list of events that are not handled in that state but, rather, are postponed and queued for handling by the object in another state An object remains in a state for a finite amount of time. For example, a Heater in a home might be in any of four states: Idle, Cooling, Heating, Initiating and Active. Transition A transition is a relationship between two states indicating that an object in the first state will perform certain actions and enter the second state when a specified event occurs and specified conditions are satisfied. Transition fires means change of state occurs. Until transition fires, the object is in the source state; after it fires, it is said to be in the target state. A transition has five parts: Source state â€” The state affected by the transition Event trigger â€” a stimulus that can trigger a source state to fire on satisfying guard condition, Guard condition â€” Boolean expression that is evaluated when the transition is triggered by the reception of the event trigger, Action â€” An executable atomic computation that may directly act on the object that owns the state machine, and indirectly on other objects that are visible to the object, Target state â€” The state that is active after the completion of the transition. Source and Target State Source State: The state that is active after the completion of the transition. Events Event is a discrete signal that happens at a point in time. It also known as a stimulus and in a kind of input to an object. Here is the characteristics of events: May cause a change in state May trigger actions â€” Actions can be internal or external May have associated conditions Signal events can be used to communicate between state machines Guard Condition State transition label â€” Event [Guard Condition] Condition is a Boolean function Conditions are optional on state machines Condition is true for finite period of time When event occurs, condition must be true for state transition to occur. If condition is false, state transition does not occur. Actions Action is executed as a result of instantaneously of state transition. Fork node is a pseudo state used to split an incoming transition into two or more transitions terminating on orthogonal target vertices. The segments outgoing from a fork vertex must not have guards or triggers and it must have exactly one incoming and at least two outgoing transitions. Join node is a pseudo state used to merge several transitions emanating from source vertices in different orthogonal regions. The transitions entering a join vertex cannot have guards or triggers and it must have at least two incoming transitions and exactly one outgoing transition. Merge node is used to bring back together different decision paths that ware created using a decision node. Choice is a pseudo state which, when reached, result in the dynamic evaluation of the guards of the triggers of its outgoing transitions. This realizes a dynamic

conditional branch. It allows splitting of transitions into multiple outgoing paths such that the decision on which path to take. Choice Node for State Machine Diagram Terminate is a pseudo state indicates that the lifeline of the state machine has ended. A terminate pseudo-state is represented by a cross. Unlike a final state, a terminate pseudo state implies that the state machine is ended due to the context object is terminated. There is no exit of any states nor does the state machine perform any exit actions other than the actions associated with the transition that leads to the terminate state. Composite State A simple state is one which has no substructure. Composite States can be further broken down into substates either within the state or in a separate diagram. A state which has substates nested states is called a composite state. Substates may be nested to any level. A nested state machine may have at most one initial state and one final state. Substates are used to simplify complex flat state machines by showing that some states are only possible within a particular context the enclosing state. Composite State vs Submachine State Besides composite state, there is another symbol called submachine state, which is semantically equivalent to a composite state. Orthogonal State A composite state with two or more regions is called orthogonal. Unlike composite states, submachine states are intended to group states, so you can reuse them. Orthogonal state is divided into two or more regions separated by a dashed line: One state of each region is always active at any point in time, i. You can use parallel and synchronized node to ordinate different substates. Concurrent Substates are independent and can complete at different time. An example of history state usage is presented in the figure below: When to draw State Machine Diagram? You can use state machines in the following situations: During business modeling, you can create state machines to model a use-case scenario. During analysis and design, you can use several state machine diagrams to show different aspects of the same state machine and its behavior. How to Draw a State Machine Diagram? A Use Case model can be developed by following the steps below. Identify entities that have complex behavior or identify a class participating in behavior whose lifecycle is to be specified Model states â€” Determine the initial and final states of the entity Model transitions Model events â€” Identify the events that affect the entity Working from the initial state, trace the impact of events and identify intermediate states Identify any entry and exit actions on the states Expand states using substates where necessary If the entity is a class, check that the action in the state are supported by the operations and relationships of the class, and if not extend the class Refine and elaborate as required You can also: Draw substates for logical categorization of states with abstraction for reuse purposes State Machine Diagram Examples The Digital Clock State Machine diagram example below shows the interface of a simple digital clock: The state machine diagram where the class it is attached: The state Diagram for modeling the behavior of the DigitalClock: Some more State Machine Diagram examples are provided below. State Machine Diagram example: Toaster State Machine Diagram example: Oven State Machine Diagram example: Computer Testing This example represents two sets of concurrent substates by using two regions. Human Life This example represents two sets of concurrent substates by using two regions. Want to draw a State Machine Diagram?

STATE TRANSITION DIAGRAM TUTORIAL pdf

3: State diagram - Wikipedia

Following is an example of a Statechart diagram where the state of Order object is analyzed The first state is an idle state from where the process starts. The next states are arrived for events like send request, confirm request, and dispatch order.

And if the user enters a wrong password he is moved to next state. If he does the same 3rd time, he will reach the account blocked state. Check this video, before you refer the example below: The Video will load in some time. If you still face issue viewing video click here In the flight reservation login screen, consider you have to enter correct agent name and password to access the flight reservation application. It gives you the access to the application with correct password and login name, but what if you entered the wrong password. The application allows three attempts, and if users enter the wrong password at 4th attempt, the system closes the application automatically. The State Graphs helps you determine valid transitions to be tested. In this case, testing with the correct password and with an incorrect password is compulsory. For the test scenarios, log-in on 2nd, 3rd and 4th attempt anyone could be tested. You can use State Table to determine invalid system transitions. In a State Table, all the valid states are listed on the left side of the table, and the events that cause them on the top. Each cell represents the state system will move to when the corresponding event occurs. For example, while in S1 state you enter a correct password you are taken to state S6 Access Granted. Suppose if you have entered the wrong password at first attempt you will be taken to state S3 or 2nd Try. Likewise, you can determine all other states. Two invalid states are highlighted using this method. Suppose you are in state S6 that is you are already logged into the application, and you open another instance of flight reservation and enter valid or invalid passwords for the same agent. System response for such a scenario needs to be tested. Advantages and Disadvantages Of State Transition Technique Advantages Disadvantages This testing technique will provide a pictorial or tabular representation of system behavior which will make the tester to cover and understand the system behavior effectively. For example, if the system is not a finite system not in sequential order , this technique cannot be used. By using this testing, technique tester can verify that all the conditions are covered, and the results are captured Another disadvantage is that you have to define all the possible states of a system. While this is all right for small systems, it soon breaks down into larger systems as there is an exponential progression in the number of states. State Transition Testing Technique is helpful where you need to test different system transitions. Two main ways to represent or design state transition, State transition diagram, and State transition table. In state transition diagram the states are shown in boxed texts, and the transition is represented by arrows. In state transition table all the states are listed on the left side, and the events are described on the top. This main advantage of this testing technique is that it will provide a pictorial or tabular representation of system behavior which will make the tester to cover and understand the system behavior efficiently.

4: Tutorial: State Machines - National Instruments

For example, in e-commerce a product will have a release or available date, a sold out state, a restocked state, placed in cart state, a saved on wish list state, a purchased state, and so on. Certain transitions will not be applicable when an object is in a particular state, for example a product can be in a purchased state or a saved in cart.

Classic state diagrams require the creation of distinct nodes for every valid combination of parameters that define the state. This can lead to a very large number of nodes and transitions between nodes for all but the simplest of systems state and transition explosion. This complexity reduces the readability of the state diagram. With Harel statecharts it is possible to model multiple cross-functional state diagrams within the statechart. Each of these cross-functional state machines can transition internally without affecting the other state machines in the statechart. The current state of each cross-functional state machine in the statechart defines the state of the system. The Harel statechart is equivalent to a state diagram but it improves the readability of the resulting diagram. Alternative semantics[edit] There are other sets of semantics available to represent state diagrams. For example, there are tools for modeling and designing logic for embedded controllers. The figure below shows a comparison of a state diagram with a flowchart. A state machine panel a performs actions in response to explicit events. In contrast, the flowchart panel b does not need explicit events but rather transitions from node to node in its graph automatically upon completion of activities. The reason is that each node in a flowchart represents a program command. A program command is an action to be executed. In more detail, the source code listing represents a program graph. Executing the program graph parsing and interpreting results in a state graph. So each program graph induces a state graph. Conversion of the program graph to its associated state graph is called "unfolding" of the program graph. The program graph is a sequence of commands. If no variables exist, then the state consists only of the program counter, which keeps track of where in the program we are during execution what is the next command to be applied. In this case before executing a command the program counter is at some position state before the command is executed. Executing the command moves the program counter to the next command. Since the program counter is the whole state, it follows that executing the command changed the state. So the command itself corresponds to a transition between the two states. Now consider the full case, when variables exist and are affected by the program commands being executed. Then between different program counter locations, not only does the program counter change, but variables might also change values, due to the commands executed. Consequently, even if we revisit some program command e. In the previous case, the program would be in the same state, because the whole state is just the program counter, so if the program counter points to the same position next command it suffices to specify that we are in the same state. The term "unfolding" originates from this multiplication of locations when producing the state graph from the program graph. A representative example is a do loop incrementing some counter until it overflows and becomes 0 again. Although the do loop executes the same increment command iteratively, so the program graph executes a cycle, in its state space is not a cycle, but a line. This results from the state being the program location here cycling combined with the counter value, which is strictly increasing until the overflow , so different states are visited in sequence, until the overflow. After the overflow the counter becomes 0 again, so the initial state is revisited in the state space, closing a cycle in the state space assuming the counter was initialized to 0. The figure above attempts to show that reversal of roles by aligning the arcs of the state diagrams with the processing stages of the flowchart. You can compare a flowchart to an assembly line in manufacturing because the flowchart describes the progression of some task from beginning to end e. A state machine generally has no notion of such a progression. A state in a state machine is an efficient way of specifying a particular behavior, rather than a stage of processing. Other extensions[edit] An interesting extension is to allow arcs to flow from any number of states to any number of states. This only makes sense if the system is allowed to be in multiple states at once, which implies that an individual state only describes a condition or other partial aspect of the overall, global state. The resulting formalism is known as a Petri net. Another extension allows the integration of flowcharts within Harel statecharts. This extension supports the development of software that is both event

driven and workflow driven.

STATE TRANSITION DIAGRAM TUTORIAL pdf

5: What is State Transition Testing? State Transition Diagram

State Transition Diagram and State Transition Table There are two main ways to represent or design state transition, State transition diagram, and state transition table. In state transition diagram the states are shown in boxed texts, and the transition is represented by arrows.

State Transition Diagrams State transition diagrams have been used right from the beginning in object-oriented modeling. The basic idea is to define a machine that has a number of states hence the term finite state machine. The machine receives events from the outside world, and each event can cause the machine to transition from one state to another. For an example, take a look at figure 1. Here the machine is a bottle in a bottling plant. It begins in the empty state. In that state it can receive squirt events. If the squirt event causes the bottle to become full, then it transitions to the full state, otherwise it stays in the empty state indicated by the transition back to its own state. When in the full state the cap event will cause it to transition to the sealed state. The diagram indicates that a full bottle does not receive squirt events, and that an empty bottle does not receive cap events. Thus you can get a good sense of what events should occur, and what effect they can have on the object. State transition diagrams were around long before object modeling. They give an explicit, even a formal definition of behavior. A big disadvantage for them is that they mean that you have to define all the possible states of a system. Whilst this is all right for small systems, it soon breaks down in larger systems as there is an exponential growth in the number of states. This state explosion problem leads to state transition diagrams becoming far too complex for much practical use. To combat this state explosion problem, object-oriented methods define separate state-transition diagrams for each class. This pretty much eliminates the explosion problem since each class is simple enough to have a comprehensible state transition diagram. It does, however, raise a problem in that it is difficult to visualize the behavior of the whole system from a number of diagrams of individual classes - which leads people to interaction and activity modeling. The most popular variety of state-transition diagram in object methods is the Harel Statechart as in Figure 1. It is one of the more powerful and flexible forms of state transition diagram. A particularly valuable feature of the approach is its ability to generalize states, which allows you to factor out common transitions thus I can show that the break event applies to both full and empty states by creating the super-state of in-progress. It also has a flexible approach to handling processing. Processes that are instantaneous i. Processes that are long and can be interrupted are bound to states, these are called activities. Transitions can also have a condition attached to them, which means that the transition only occurs if the condition is true. There is also a capability for concurrent state diagrams, allowing objects to have more than one diagram to describe their behavior. A Harel statechart Not all methods use Harel Statecharts. This form only allows processes to occur when in a state hence the extra state in Figure 8 and has no superstates. This is a good example of the question of expressiveness in techniques. The Harel Statechart is clearly a more expressive technique, but since it is more expressive there is more to learn when using it. In addition, it is more difficult to implement. It is the equivalent to figure 1. When to Use Them State models are ideal for describing the behavior of a single object. They are also formal, so tools can be built which can execute them. Their biggest limitation is that they are not good at describing behavior that involved several objects, for these cases use an interaction diagram or an activity diagram. People often do not find drawing state diagrams for several objects to be a natural way of describing a process. In these cases you can try either drawing a single state diagram for the process, or using an activity diagram. This defines the basic behavior, which you then need to refactor to split it across a number of objects. For an initial tutorial on them I would suggest either [Booch] or [Rumbaugh]. For a more in-depth treatment take a look at [Cook and Daniels]; they give a lot of good details on formalisms, an integration of design by contract, and a discussion of the use of state diagrams with subclassing.

STATE TRANSITION DIAGRAM TUTORIAL pdf

6: State Transition Testing

The following diagram gives an example where the state executed after the main processing state depends on which route is used to transition out of the state. Choice Pseudo-State A choice pseudo-state is shown as a diamond with one transition arriving and two or more transitions leaving.

AgileModeling Objects have both behavior and state or, in other words, they do things and they know things. Some objects do and know more things, or at least more complicated things, than other objects. Some objects are incredibly complicated, so complex that developers can have difficulty understanding them. To understand complex classes better, particularly those that act in different manners depending on their state, you should develop one or more UML 2 state machine diagrams, formerly called state chart diagrams in UML 1. UML state machine diagrams depict the various states that an object may be in and the transitions between those states. In fact, in other modeling languages, it is common for this type of a diagram to be called a state-transition diagram or even simply a state diagram. A state represents a stage in the behavior pattern of an object, and like UML activity diagrams it is possible to have initial states and final states. An initial state, also called a creation state, is the one that an object is in when it is first created, whereas a final state is one in which no transitions lead out of. A transition is a progression from one state to another and will be triggered by an event that is either internal or external to the object. Figure 1 presents an example state machine diagram for the Seminar class during registration. The rounded rectangles represent states: An object starts in an initial state, represented by the closed circle, and can end up in a final state, represented by the bordered circle. A seminar during registration. The arrows in Figure 1 represent transitions, progressions from one state to another. For example, when a seminar is in the Scheduled state, it can either be opened for enrollment or cancelled. It is mandatory to indicate the event which causes the transition, such as open or cancelled. Guard, conditions that must be true for the transition to be triggered, are optionally indicated. The [not seat available] guard is shown on the student enrolled transition from the Open For Enrollment to the Closed To Enrollment state. The invocation of methods, such as addToWaitingList can optionally be indicated on transitions. The order in the listing implying the order in which they are invoked. States are represented by the values of the attributes of an object. For example, a seminar is in the Open For Enrollment state when it has been flagged as open and seats are available to be filled. It is possible to indicate the invocation of methods within a state, for example, upon entry into the Closed To Enrollment state the method notifyInstructor is invoked. The notation used within the same as that used on transitions, the only difference being that the method list is mandatory and the event is optional. For example in the Full state the operations addToWaitingList and considerSplit are invoked whenever a student is enrolled. Had there been no event indicated those methods would be invoked continuously in a loop whenever the object is in that state. I indicate the methods to run during the state when I want to indicate a method is to be run continuously, perhaps a method that polls other objects for information or a method that implements the logic of an important business process. Methods to be invoked when the object enters the state are indicated by the keyword entry, as you see with both the Open For Enrollment and Closed To Enrollment states in Figure 1. Methods to be invoked as the object exits the state are indicated by the keyword exit. The capability to indicate method invocations when you enter and exit a state is useful because it enables you to avoid documenting the same method several times on each of the transitions that enter or exit the state, respectively. Transitions are the result of the invocation of a method that causes an important change in state. Understanding that not all method invocations will result in transitions is important. Furthermore, Figure 1 indicates an attempt to enroll a student in a full seminar may not result in the object changing state, unless it is determined that the seminar should be split, even though the state of the object changes another student is added to the waiting list. You can see that transitions are a reflection of your business rules. For example, you see that you can attempt to enroll a student in a course only when it is open for enrollment or full, and that a seminar may be split presumably into two seminars when the waiting list is long enough to justify the split. You can have recursive transitions, also called self transitions, that start and end in the same state. An example of which is the student dropped transition when the seminar is full. For the

sake of convention, we say an object is always in one and only one state, implying transitions are instantaneous. Although we know this is not completely true every method is going to take some time to run , this makes life a lot easier for us to assume transitions take no time to complete. Because the lifecycle of a seminar is so complex Figure 1 only depicts part of it. Figure 2 depicts the entire lifecycle, with Figure 1 shown as a substate of the Enrollment state. I could have included all of the details in Figure 2 but chose not to in order to keep the diagram simple - I prefer to follow the AM practices Depict Models Simply and Model in Small Increments. In fact, instead of creating a diagram such as Figure 2 I typically prefer something more along the lines of the high-level view of Figure 3 with detailed views such as Figure 1. This approach keeps the diagrams small and easy to understand. Top-level state machine diagram. Figure 4 depicts a slightly different take on state machine diagrams, this time it is much closer to an analysis level diagram because it shows what is happening to the seminar while it is in this state from the point of view of the people involved. It is organized into two parallel swimlanes representing parallel substates - one from the point of view of the professor teaching the seminar and the other showing the actions of the teaching assistant responsible for keeping the seminar material up to date. Concurrent substates are common with hardware but very uncommon in business classes, hence the goofy example. Normally I would depict this sort of information using either a UML activity diagram or a UML timing diagram but I needed an example to show you extra notation. The Being Taught state. Figure 4 shows several ways to depict transitions. The Break Starts transition exiting from the Being Taught states is applicable to all of the substates, you know this because it exits from the superstate instead of an individual substate. The Work Submitted transition is potentially triggered by several sources, you know this because it is attached to the outside edge of the superstate, whereas the source of the Break ends transition is explicitly defined as the School Break state. The initial transition into this state is the Term Started transition, indicated through the use of an initial state symbol. I could also have modeled this state coming from an Enrollment state, either approach is fair. The Term Started and Break Ends transitions are first merged, then they lead to a fork which in turn leads to one or the other set of concurrent substates. A history pseudo-state is shown, the circle with the H, indicating that if Seminar was previously in this state, left it, and the returns that it will go back to the substate it was originally in. The arrow leaving the history pseudo state indicates that the Deliver Course Material substate is the default the very first time that Seminar enters the Begin Taught superstate. Drawing State Machine Diagrams When drawing a state machine diagram the thing you want to do is to identify the creation state and whether any final states exist. After you have done this, ask yourself what other states or stages in the life of an object does it pass through? You can often find states by looking at the boundary values of your attributes. For example, when the number of students in a seminar reaches the maximum, it becomes full. Full is a valid state because different rules now apply: Once you have identified as many states as you can, start looking for transitions. For each state, ask yourself how the object can get out of it, if possible. This will give you a transition. You should also look at the methods you identified in your class diagram. Some of them will correspond to a transition in your state diagram. When the answer is no, you may need to document this potential issue so your programmers develop the proper error checking code, so the transition is not allowed to occur. Although being able to inherit state diagrams would be nice, it is extremely unlikely this will happen. The definition of inheritance says that although the subclass is similar to the superclass, it is still different. The behavior of the subclass is usually different than that of the superclass. This means you need to reconsider the state diagram when you inherit from a class with one. The one good thing is many of the states and transitions are reusable. You will probably find you either add new states and transitions, or you will redefine some. Remaining Agile State machine modeling is a dynamic modeling technique, one that focuses on identifying the behavior within your system-in this case, behavior specific to the instances of a single class. My style is to draw one or more state machine diagrams when a class exhibits different behavior depending on its state. For example, the Address class is fairly simple, representing data you will display and manipulate in your system. Seminar objects, on the other hand, are fairly complex, and therefore it makes sense to create a state machine diagram for them. However, state machine diagrams are much more common in real-time systems Douglass

STATE TRANSITION DIAGRAM TUTORIAL pdf

7: State Machine Diagram - UML Diagrams - Unified Modeling Language Tool

State transition diagrams have been used right from the beginning in object-oriented modeling. The basic idea is to define a machine that has a number of states (hence the term finite state machine). The machine receives events from the outside world, and each event can cause the machine to.

State Definition A state models a situation during which some usually implicit invariant condition holds. The invariant may represent a static situation such as an object waiting for some external event to occur. However, it can also model dynamic conditions such as the process of performing some behavior i. Properties The name of state. Entry An optional behavior that is executed whenever this state is entered regardless of the transition taken to reach the state. If defined, entry actions are always executed to completion prior to any internal behavior or transitions performed within the state. Exit An optional behavior that is executed whenever this state is exited regardless of which transition was taken out of the state. If defined, exit actions are always executed to completion only after all internal activities and transition actions have completed execution. Do activity An optional behavior that is executed while being in the state. The execution starts when this state is entered, and stops either by itself or when the state is exited whichever comes first. State invariant Specifies conditions that are always true when this state is the current state. In protocol state machines, state invariants are additional conditions to the preconditions of the outgoing transitions, and to the postcondition of the incoming transitions. Redefined state The state of which this state is a redefinition. Documentation Description of state. Regions A region is an orthogonal part of either a composite state or a state machine. It contains states and transitions. Deferrable Triggers A list of triggers that are candidates to be retained by the state machine if they trigger no transitions out of the state not consumed. A deferred trigger is retained until the state machine reaches a state configuration where it is no longer deferred. Submachine State Definition A submachine state specifies the insertion of the specification of a submachine state machine. The state machine that contains the submachine state is called the containing state machine. The same state machine may be a submachine more than once in the context of a single containing state machine. A submachine state is semantically equivalent to a composite state. The regions of the submachine state machine are the regions of the composite state. The entry, exit, and behavior actions and internal transitions are defined as part of the state. Submachine state is a decomposition mechanism that allows factoring of common behaviors and their reuse.

STATE TRANSITION DIAGRAM TUTORIAL pdf

8: How to draw a state machine diagram

State Transition Diagram Example - Georgia Tech - Software Development Process State Transition Testing UML Use Case Diagram Tutorial - Duration: Lucidchart , views.

Next Page The name of the diagram itself clarifies the purpose of the diagram and other details. It describes different states of a component in a system. A Statechart diagram describes a state machine. State machine can be defined as a machine which defines different states of an object and these states are controlled by external or internal events. Activity diagram explained in the next chapter, is a special kind of a Statechart diagram. As Statechart diagram defines the states, it is used to model the lifetime of an object. Purpose of Statechart Diagrams Statechart diagram is one of the five UML diagrams used to model the dynamic nature of a system. They define different states of an object during its lifetime and these states are changed by events. Statechart diagrams are useful to model the reactive systems. Reactive systems can be defined as a system that responds to external or internal events. Statechart diagram describes the flow of control from one state to another state. States are defined as a condition in which an object exists and it changes when some event is triggered. The most important purpose of Statechart diagram is to model lifetime of an object from creation to termination. Statechart diagrams are also used for forward and reverse engineering of a system. However, the main purpose is to model the reactive system. To model the life time of a reactive system. To describe different states of an object during its life time. Define a state machine to model the states of an object. How to Draw a Statechart Diagram? Statechart diagram is used to describe the states of different objects in its life cycle. Emphasis is placed on the state changes upon some internal or external events. These states of objects are important to analyze and implement them accurately. Statechart diagrams are very important for describing the states. States can be identified as the condition of objects when a particular event occurs. Following is an example of a Statechart diagram where the state of Order object is analyzed The first state is an idle state from where the process starts. The next states are arrived for events like send request, confirm request, and dispatch order. These events are responsible for the state changes of order object. During the life cycle of an object here order object it goes through the following states and there may be some abnormal exits. This abnormal exit may occur due to some problem in the system. When the entire life cycle is complete, it is considered as a complete transaction as shown in the following figure. The initial and final state of an object is also shown in the following figure. Where to Use Statechart Diagrams? From the above discussion, we can define the practical applications of a Statechart diagram. Statechart diagrams are used to model the dynamic aspect of a system like other four diagrams discussed in this tutorial. However, it has some distinguishing characteristics for modeling the dynamic nature. Statechart diagram defines the states of a component and these state changes are dynamic in nature. Its specific purpose is to define the state changes triggered by events. Events are internal or external factors influencing the system. Statechart diagrams are used to model the states and also the events operating on the system. When implementing a system, it is very important to clarify different states of an object during its life time and Statechart diagrams are used for this purpose. When these states and events are identified, they are used to model it and these models are used during the implementation of the system. If we look into the practical implementation of Statechart diagram, then it is mainly used to analyze the object states influenced by events. This analysis is helpful to understand the system behavior during its execution. To model the reactive system. Reactive system consists of reactive objects. To identify the events responsible for state changes. Forward and reverse engineering.

STATE TRANSITION DIAGRAM TUTORIAL pdf

9: State Machine Diagram - UML 2 Tutorial | Sparx Systems

A state diagram is a type of diagram used in computer science and related fields to describe the behavior of systems. State diagrams require that the system described is composed of a finite number of states; sometimes, this is indeed the case, while at other times this is a reasonable abstraction.

The term connection-oriented means the two applications using TCP must establish a TCP connection with each other before they can exchange data. It is a full duplex protocol, meaning that each TCP connection supports a pair of byte streams, one flowing in each direction. TCP includes a flow-control mechanism for each of these byte streams that allows the receiver to limit how much data the sender can transmit. TCP also implements a congestion-control mechanism. Two processes communicating via TCP sockets. TCP provides the following facilities to: TCP itself decides how to segment the data and it may forward the data at its own convenience. If the ACK is not received within a timeout interval, the data is retransmitted. The receiving TCP uses the sequence numbers to rearrange the segments when they arrive out of order, and to eliminate duplicate segments. Flow Control The receiving TCP, when sending an ACK back to the sender, also indicates to the sender the number of bytes it can receive beyond the last received TCP segment, without causing overrun and overflow in its internal buffers. This is sent in the ACK in the form of the highest sequence number it can receive without problems. Multiplexing To allow for many processes within a single host to use TCP communication facilities simultaneously, the TCP provides a set of addresses or ports within each host. Concatenated with the network and host addresses from the internet communication layer, this forms a socket. A pair of sockets uniquely identifies each connection. Logical Connections The reliability and flow control mechanisms described above require that TCP initializes and maintains certain status information for each data stream. The combination of this status, including sockets, sequence numbers and window sizes, is called a logical connection. Each connection is uniquely identified by the pair of sockets used by the sending and receiving processes. Full Duplex TCP provides for concurrent data streams in both directions. The figure shows the format of the TCP header. Its normal size is 20 bytes unless options are present. Each of the fields is discussed below: The SrcPort and DstPort fields identify the source and destination ports,respectively. These two fields plus the source and destination IP addresses, combine to uniquely identify each TCP connection. The sequence number identifies the byte in the stream of data from the sending TCP to the receiving TCP that the first byte of data in this segment represents. The Acknowledgement number field contains the next sequence number that the sender of the acknowledgement expects to receive. This is therefore the sequence number plus 1 of the last successfully received byte of data. This field is valid only if the ACK flag is on. Once a connection is established the Ack flag is always on. The Acknowledgement and AdvertisedW indow fields carry information about the flow of dat going in the other direction. This is done using the AdvertisedWindow field. The sender is then limited to having no more than a value of AdvertisedWindow bytes of un acknowledged data at any given time. The receiver sets a suitable value for the AdvertisedWindow based on the amount of memory allocated to the connection for the purpose of buffering data. The header length gives the length of the header in bit words. This is required because the length of the options field is variable. The 6-bit Flags field is used to relay control information between TCP peers. The ACK flag is set any time the Acknowledgement field is valid, implying that the receiver should pay attention to it. The URG flag signifies that this segment contains urgent data. When this flag is set, the UrgPtr field indicates where the non-urgent data contained in this segment begins. The PUSH flag signifies that the sender invoked the push operation, whic h indicates to the receiving side of TCP that it should notify the receiving process of this fact. Finally, the RESET flag signifies that the receiver has become confused and so wants to abort the connection. The Checksum covers the TCP segment: This is a mandatory field that must be calculated by the sender, and then verified by the receiver. The Option field is the maximum segment size option, called the MSS. Each end of the connection normally specifies this option on the first segment exchanged. It specifies the maximum sized segment the sender wants to recieve. The data portion of the TCP segment is optional. In the process of terminating a connection, the important thing to kee p in mind is that the application process on both sides of

the connection must independently close its half of the connection. This side closes first: The other side closes first: Both sides close at the same time: As a consequence this other side might re transmit its FIN segment, and this second FIN segment might be delayed in the network. If the connection were allowed to move directly to the CLOSED state, then another pair of application processes might come along and open the same connection, and the delayed FIN segment from the earlier incarnation of the connection would immediately initiate the termination of the later incarnation of that connection. Reliable and ordered delivery The sending and receiving sides of TCP interact in the following manner to implement reliable and ordered delivery: Each byte has a sequence number.

STATE TRANSITION DIAGRAM TUTORIAL pdf

Whos that in the itsy-bitsy, anyway? Stevi Mittman The father of British Canada 2 Enter the Peacock Blue 22 Emotionally Handicapped Children (Teachers License Examination Ser, T-69) Wilson, E.M. On the Romance que dize mi padre era de Ronda. Organic harmonies, 1855-1862 Mitsubishi galant service manual Alterity and Facticity New Perspectives on Husserl (Phaenomenologica) The structure and content of early representational play : the case of building blocks Stuart Reifel Challenges in construction project management Developing creativity in gifted students The effects of ICT standards on educational motivation and learning : the Australian experience Juhani E. Enriching our worship Communication Technologies Challenge of social equality Advances in Electrochemical Science and Engineering (Advances in Electrochemical Sciences and Engineering Fe review manual 3rd edition The Negro American Family The Negro American Artisan Efforts for Social Betterment Among Negro Americans The structure of affine buildings Qualitative research results and discussion KISS Guide to Wine Blue Beach and Oharan Kanoa p. 58 In the company of writers The Hong Kong dream Brian Lawless Legal research guide to television broadcasting and program syndication Two of the Deadliest Preface to a modern mythology 16 Strengthening provincial and local planning and expenditure management What is dramatherapy? Organic reactive intermediates Jack And the Beanstalk (Young Reading Gift Books) Agroforestry practices in nigeria Saintly workers, 5 lectures Catholic Church of Macon City, Mo. Walnuts A Medical Dictionary, Bibliography, and Annotated Research Guide to Internet References California focus on earth science workbook One Hundred Years of Phenomenology Tough on crime or tough on the causes of crime? St. Petersburg City Streets Map Scratch the Boatyard Cat