Software Construction, Renovation,
& Maintenance

The State of State

Jim Ladd

November 9, 2024

https://www.linkedin.com/pulse/state-jim-ladd-ssgke

Object-oriented programming has now been around for over five decades, yet it feels like just yesterday that my colleagues and I were diving into object-oriented analysis and design (OOAD) methods. Back in 1989, we found ourselves debating whether an object should be "this big" (arms spread wide) or "this big" (thumb and index finger barely separated). It sounds amusing now, but at the time, these questions demanded thorough research, deep discussions, and repeated prototyping as we carved out a new path toward understanding.

> *"An object is an entity that has state, behavior, and identity."*
>
> Grady Booch - Object Oriented Design with Applications

Of the three characteristics of an object, state can be the most complex and, commonly, is the least understood. Before continuing, let's defer to Grady Booch again for a definition:

> *"The state of an object encompasses all of the (usually static) properties of the object plus the current (usually dynamic) values of each of these properties."*

The first OOAD technique my team adopted was the Shlaer-Mellor Method by Sally Shlaer and Stephen J. Mellor. This method focuses on partitioning the problem and solution spaces, conducting domain analysis, and employing recursive design. A key feature is the assumption that all objects and relationships have lifecycles. (Shlaer & Mellor, 1989) A state model is created for each object, capturing their dynamic behaviors. These state models are initially constructed in the problem space and can later be translated into various forms within the solution space.

Throughout my career, I have extensively utilized state models to analyze problem spaces. From these models, I have implemented a wide range of solutions, from simple attributes with alphanumeric values to complex design patterns and even concurrent finite state machine engines. (McCarter & Ladd, 2008) (Ladd & Alvey, Putting Physhun To Work, 2008)

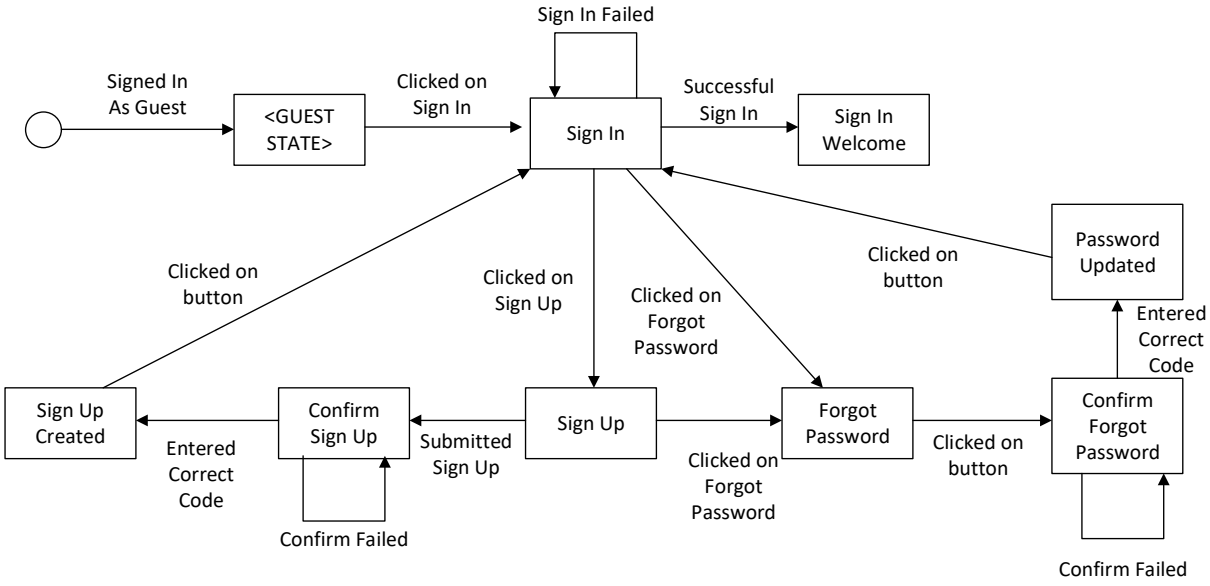
However, in recent years, attention to state modeling has diminished, and this practice has become rare. I have noticed that, from computer science interns to seasoned software developers, the understanding, appreciation, and application of state modeling are nearly absent. I strongly believe that consideration of state is as vital now as it has ever been. This

article demonstrates the significance of state and how state modeling can be effectively applied in real-world scenarios.

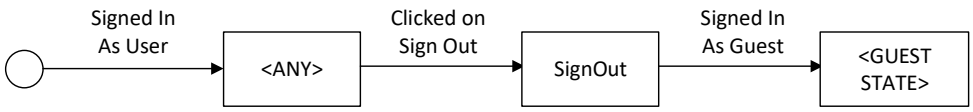
New Construction

At Wazee Group, I am currently working on a system that leverages state models within the problem space. In this system, when a user initially interacts with the web app, she is automatically signed in as an AWS Cognito Guest. As the user progresses through the process, she is prompted to register and log in to continue with the business flow. The login and registration process includes standard features like confirmation, password recovery, and password maintenance.

To streamline the development of the web app, I created a state model for the login process. This model has proven invaluable in facilitating both coding and testing activities, helping ensure a smooth and reliable user experience.



The sign out process is a much similar state model as shown below.

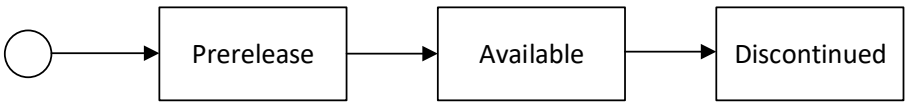


Currently the state models are used for the analysis efforts and not the implementation. I'm considering integrating FSMs in the Apache Flex architecture like the way I did with React and Redux a few years ago. (Ladd, Integrating Finite State Models with React and Redux, 2020)

Renovation

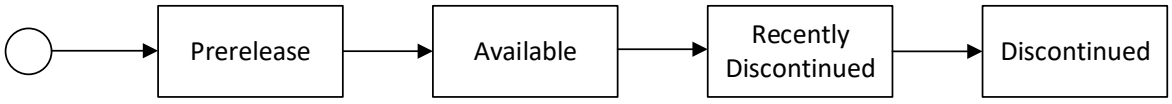
A client recently reached out with an enhancement request that was an ideal fit for a state-based technique. At the core of this client's ecosystem is an enterprise resource planning (ERP) system. The client has developed a suite of web services that embed the ERP's API, facilitating the interactions between the ERP and other systems across the enterprise. For instance, a single endpoint for entering an order into the ERP system encapsulates a complex process with over 20 API calls.

The ERP system includes different state values for its key business objects, such as orders, products, and shipments. Within the ERP, lifecycles of the core business objects are represented as a sequence of ascending numeric values, with each transition moving from a lower to a higher number. Below is a simplified state model for a product offered by the client, where the numeric values have been replaced with descriptive names to enhance readability.



The order entry systems periodically request the inventory of "available" products from the inventory service. Upon receiving this request, the inventory service queries the product service for a list of products currently in the "Available" state. The inventory service then checks the ERP system for the current stock of each product. Finally, all this data is returned to the system that initiated the request, ensuring accurate and up-to-date inventory information.

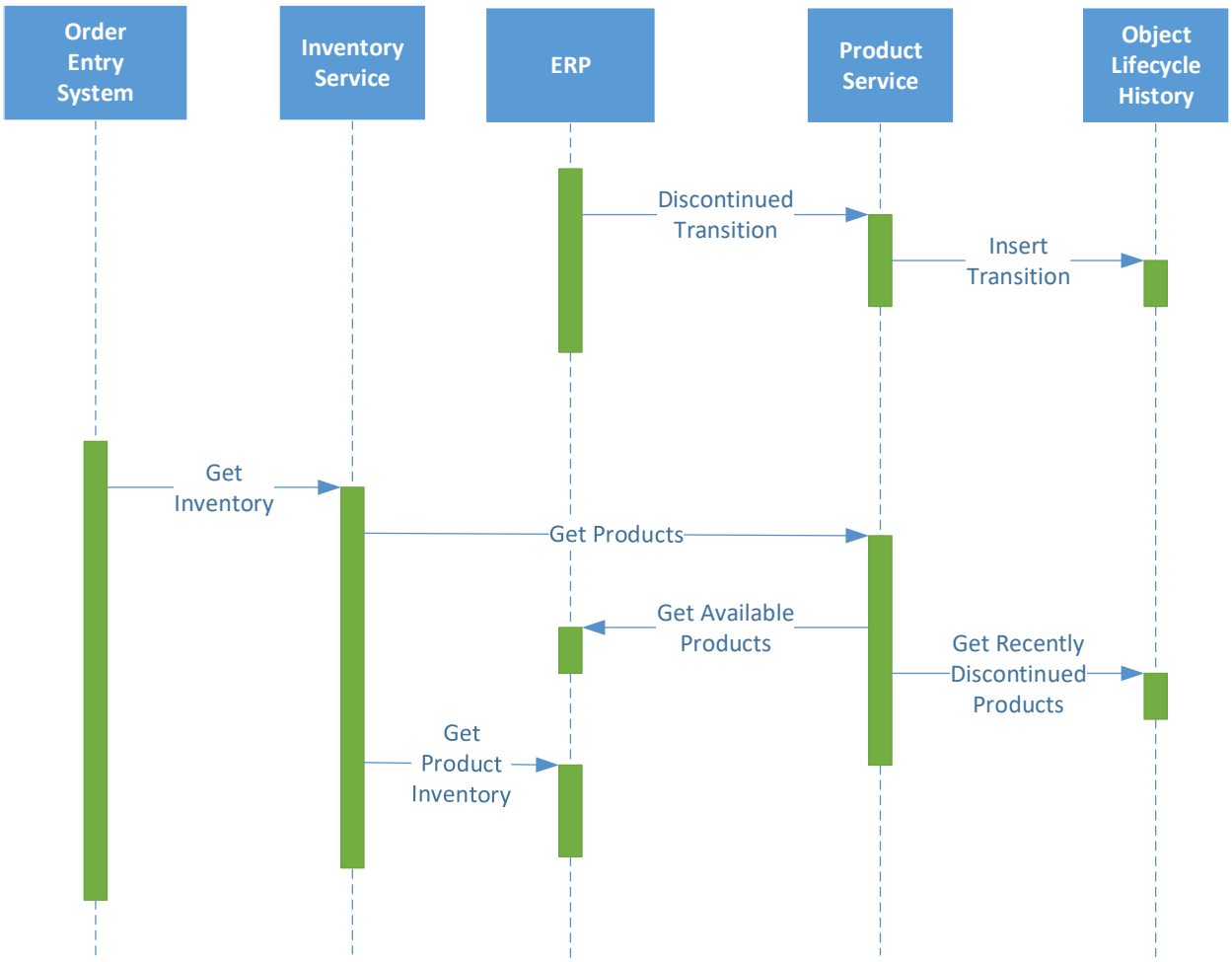
The order entry systems wanted to improve this process by including the set of products that were "recently" discontinued, i.e. those products whose state was set to "Discontinued" within the past two weeks. The "implicit" state model for the product was modified to the following:



One approach involves running a batch program that queries the ERP system for "Discontinued" objects where the "Last Modified Date" is less than two weeks. However, this design is suboptimal because the Last Modified Date may not accurately reflect the timestamp of the state change. Any updates to the product record, regardless of their relevance to its discontinuation status, would refresh the timestamp, potentially causing a very old product to be incorrectly marked as "Recently Discontinued."

A more effective approach was to capture the state transition from "Available" to "Discontinued" and persist it as a two-week "Recently Discontinued" virtual state, as depicted in the diagram. To track this transition, we utilized a feature in the ERP system that publishes messages when specific data changes occur in its internal database. Since the ERP system was already broadcasting a message whenever a product transitioned from "Available" to "Discontinued," we created a listener to receive this message and invoke a new endpoint in the product web service.

The software for this endpoint inserts relevant data, including the object type ("PRODUCT"), object ID ("111-222-333"), state ("DISCONTINUED"), and the current timestamp into a datastore. When the inventory of available products is requested, the product web service retrieves the set from the ERP system as usual, then queries the "Object Lifecycle History" datastore to add any products in the "Recently Discontinued" state to the list. A batch program runs daily to remove data older than two weeks from the datastore. An event sequence diagram illustrating these processes is shown below:

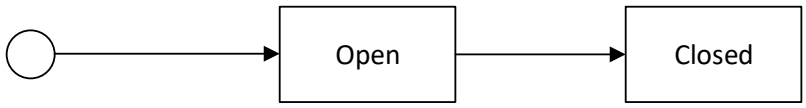


Maintenance

State modeling can also play a valuable role in software maintenance. In a recent client engagement, we encountered a recurring error that required significant manual intervention to resolve. This issue involved a global shipping provider with a unique approach to streamlining customs processing for international shipments. Their program allows multiple packages to be consolidated into a large container, which is then shipped to the destination country. The entire container passes through customs as a single unit, simplifying the process. Once it reaches the shipper's distribution center in the destination country, the packages are removed from the container and shipped domestically.

The client's software for this domain has two primary objects, Container and Package. At the beginning of a business day, an operator in the client's warehouse would create an instance of the Container object with an "Open" state. Throughout the next twenty-four hours, any shipment (i.e. Package) that was being shipped to the target country would be

assigned to the Container in the “Open” state. At the end of the day, an operator would “close” the container, the software would validate the Packages (some may have been canceled during the shift) and reconcile the Packages against their assigned Container. Once this process was completed, the software would update the Container object with the “Closed” state. The operator would then create a new Container instance (in the “Open” state) for the upcoming shift. The state model for the Container is shown below:

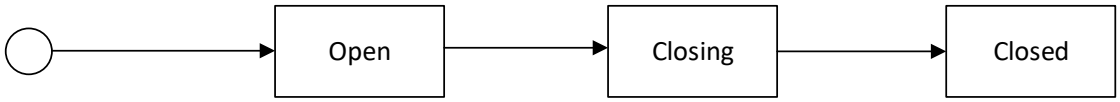


The Package object did not have a state attribute, but its implicit state model is depicted in this diagram:

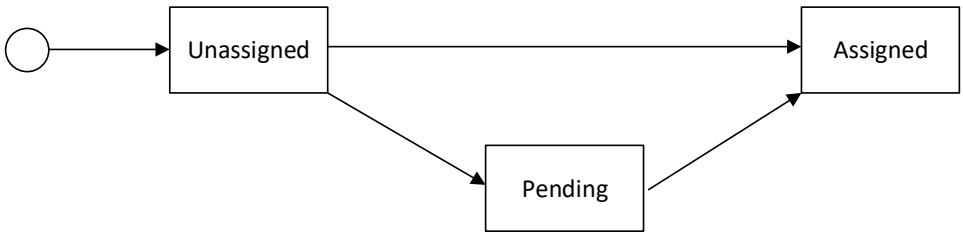
As shipment volumes increased to hundreds per day, the client began to encounter irregular exceptions during the period when a Container was being closed. After investigating the source code and reviewing log files, we identified the root cause of the issue.

The problem occurred during the interval between closing the open Container instance and creating a new one. As the volume of shipments grew, the duration of the closing process extended to several seconds. During this window when no open Container existed, a new Package would attempt to be assigned. Since there was no open Container available at that moment, an exception was triggered, necessitating manual intervention to correct the error.

A quick and effective solution was to modify the state model for the Container by introducing a new state called "Closing." This adjustment involved transitioning the state of the Container from "Open" to "Closing" at the beginning of the reconciliation process. Once this process was completed, the state would then transition to "Closed." The updated state model is illustrated in the diagram below:



The state model for the Package was made explicit with the addition of a state attribute. When a new instance of the Package object is created, it is initially assigned the "Unassigned" state. During the assignment process, the software scans for an available Container with an "Open" state. If no such Container exists, the Package is transitioned to the "Pending" state. However, if the assignment can be completed, the Package is transitioned to the "Assigned" state. The updated state model for a Package is illustrated in the diagram below:



When a new Container is created and transitions to the Open state, the software was modified so that it scans the set of Package instances in the Pending state. Those instances are assigned to the Container and are transitioned to the Assigned state.

References

Ladd, J. (2020, June 29). *Integrating Finite State Models with React and Redux*. Retrieved from SOFWERX: https://jladd413.github.io/docs/ReactReduxFSM.pdf

Ladd, J., & Alvey, T. (2008, August 1). *Putting Physhun To Work*. Retrieved from The Server Side: https://jladd413.github.io/docs/PuttingPhyshunToWork.pdf

McCarter, J., & Ladd, J. (2008, July 18). *Implementing Finite State Machines with Physhun and Spring*. Retrieved from The Server Side: https://jladd413.github.io/docs/ImplementingFiniteStateMachinesPhyshunSpring.pdf

Shlaer, S., & Mellor, S. J. (1989). Object-Oriented Approach to Domain Analysis. *ACM SIGSOFT - Software Engineering Notes Vol 14 no 5*, 70.