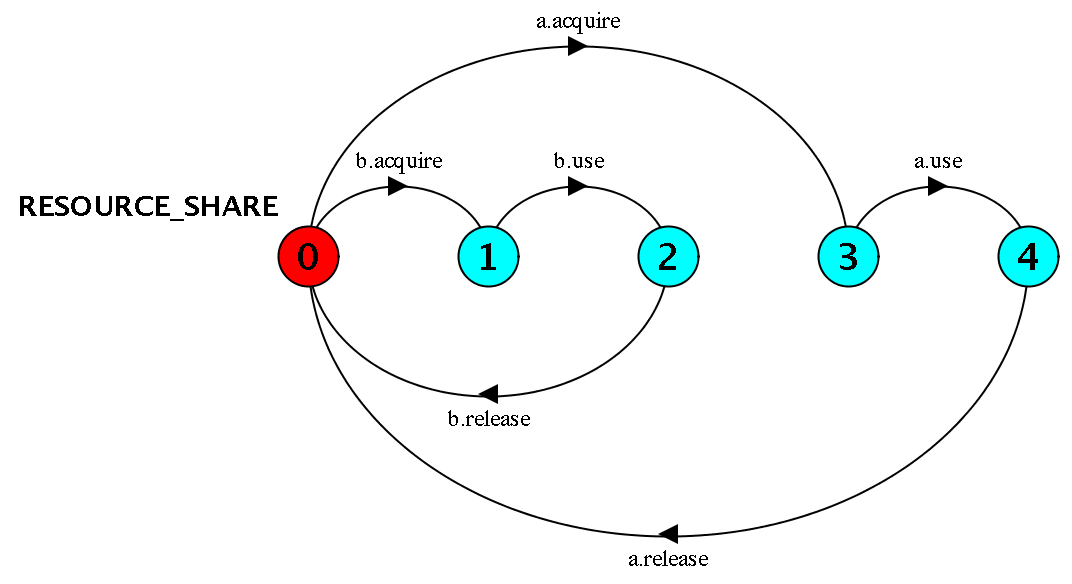
In the early stage of our project, we did not consider adopting concurrency in our implementation due to the small user size. However, during the testing, we discovered some issue might happen due to a non-concurrent design. For example, if one client is querying a resource while another client is removing it at the same time, the query can have two results, one being query successful due to the query happened before removing, other is the command returns resource not exist because of the server processed remove command earlier. As said in the project specification, this issue may not occur when the system is small. But if there is hundreds of client, the clash of commands as the example above will have more change to happen. In another word, the behaviour that two clients are using the same resource should be forbidden. In the figure is a labelled transition system of a resource share system(modified from the source code written by Jeff Kramer, Chapter 3 Concurrent Execution, Concurrency lecture notes Imperial College 2014), which is the desired resource share process when accessing by two clients.



With that in mind, we discussed the possibility of adding a lock in the system. The lock should be locked whenever some client is using the resource, then unlock and notify any client waiting after using it. The lock should be an ideal method for a situation like this. However, our implementation made it hard to achieve. We are using a hash map as our resource list, and locking the resource list makes all access of the list pause when a client is using it, even the other clients are wishing to access another resource in the list, and the lock was difficult to add to the resource itself. We think if the protocol forbids the action that two clients are accessing the same resource list at the same time, this issue will be solved easily, or another way to store the resources such as a database would also help with the problem.