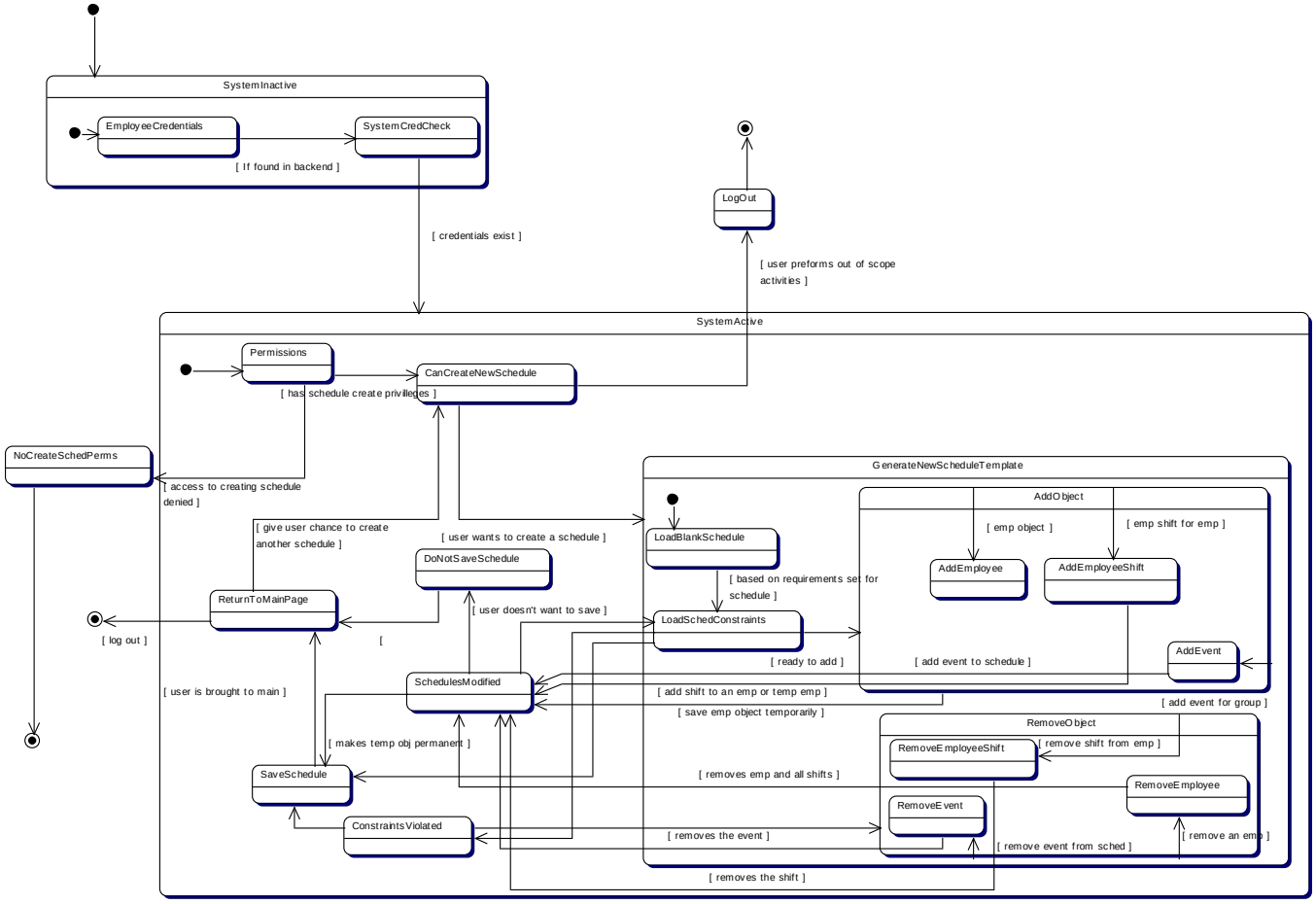
Create New Schedule



State Detail

EndState *EndState1*

User does not have permissions to create a schedule, so the state becomes an inactive state (where user cannot perform actions in the scope of this diagram) even though the user is not logged out of the system. Essentially the system is left in a semi-active state until user logs off.

StartState *existState*

This state is how the system is before any modified states are introduced. Theoretically, it represents an idle server containing the system. An inactive system implies that a user is attempting to use it.

EndState *finishedWorkState*

User has finished activities in the system and returns the system to an inactive state, or one before they logged on.

EndState *loggedOutState*

User has requested that the system return to an inactive state. From there, system can adopt the 'not-in-use' state where system on the server is idle and has no connections.

State *LogOut*

System begins to return to an inactive state.

State *NoCreateSchedPerms*

The state of permissions is absent and the system cannot deal with scheduling.

State *SystemActive*

The system is now active and supports different smaller states of creating and modifying a schedule. Deleting is not covered, but would be part of modifying the schedule.

Substates

State *CanCreateNewSchedule*

System state allows for creation of schedules.

State *ConstraintsViolated*

If objects in the schedule are illegal, the schedule will refuse to save the schedule until the constraints are met.

State *DoNotSaveSchedule*

System enters a state where it is free of temporary objects; any that are present are discarded.

State *GenerateNewScheduleTemplate*

When a schedule is generated, the state now supports a temporary object that is modified before it is officialized into the backend.

Substates

State *AddObject*

Temporary object can be modified by adding objects to it. Will automatically change state to an object-modified state that can then process the new temporary object state

Substates

State *AddEmployee*

System creates a temporary employee object that can be modified; state changed to deal with unsaved temporary objects.

State *AddEmployeeShift*

Shift is added to a temporary employee object; state changed to deal with unsaved temporary objects.

State *AddEvent*

An event is added to the schedule. This can be a meeting, special holiday event, or request (etc). state changed to deal with unsaved temporary objects.

StartState *blankScheduleExistsState*

This state monitors actions relating to the modification of a temporary object that represents a blank schedule object.

State *LoadBlankSchedule*

Temporary object is initialized and stored in memory.

State *LoadSchedConstraints*

Loads constraints that are found in the system, checks all current temporary data against them, and will allow for modification or block illegal modifications.

State *RemoveObject*

An object is removed from the temp object, state is changed as schedule is modified.

Substates

State *RemoveEmployee*

A temporary object has been removed, causing state to change and deal with modified temporary objects.

State *RemoveEmployeeShift*

A shift from a temporary employee is removed (if exists), causing state to change and deal with modified temporary objects.

State *RemoveEvent*

An event is removed (if exists), causing state to change and deal with modified temporary objects.

State *Permissions*

System is in a state of checking permissions

State *ReturnToMainPage*

Returns to the basic active state where user can choose to create a new schedule or log out.

State *SaveSchedule*

The schedule is saved to the backend and the state is reverted so that the currently loaded object (temp) cannot be saved unless another modification is made.

State *SchedulesModified*

The schedule has been modified, so attempting to exit will prompt a save dialog before logging out. This also invokes a state where the user can save the temporary object.

StartState *systemActiveState*

The system now has a legitimate connection to a user. The state now must accept action requests from the user.

State *SystemInactive*

This state represents the system status to the user; since the user does not have an active session running, the system is inactive.

Substates

State *EmployeeCredentials*

The system state is still inactive when employee credentials (login and password) are entered.

StartState *inactiveSystemstate*

The system is in a state where no actions are being performed, but a user is 'observing' the system and the system will respond to requests

State *SystemCredCheck*

The system checks the entered credentials against a list of acceptable ones, changing the system state if they are found.