Team
Blue Steel

Design Patterns and Rationale

Design Pattern 1: *Façade Pattern*

Design problem to be solved:

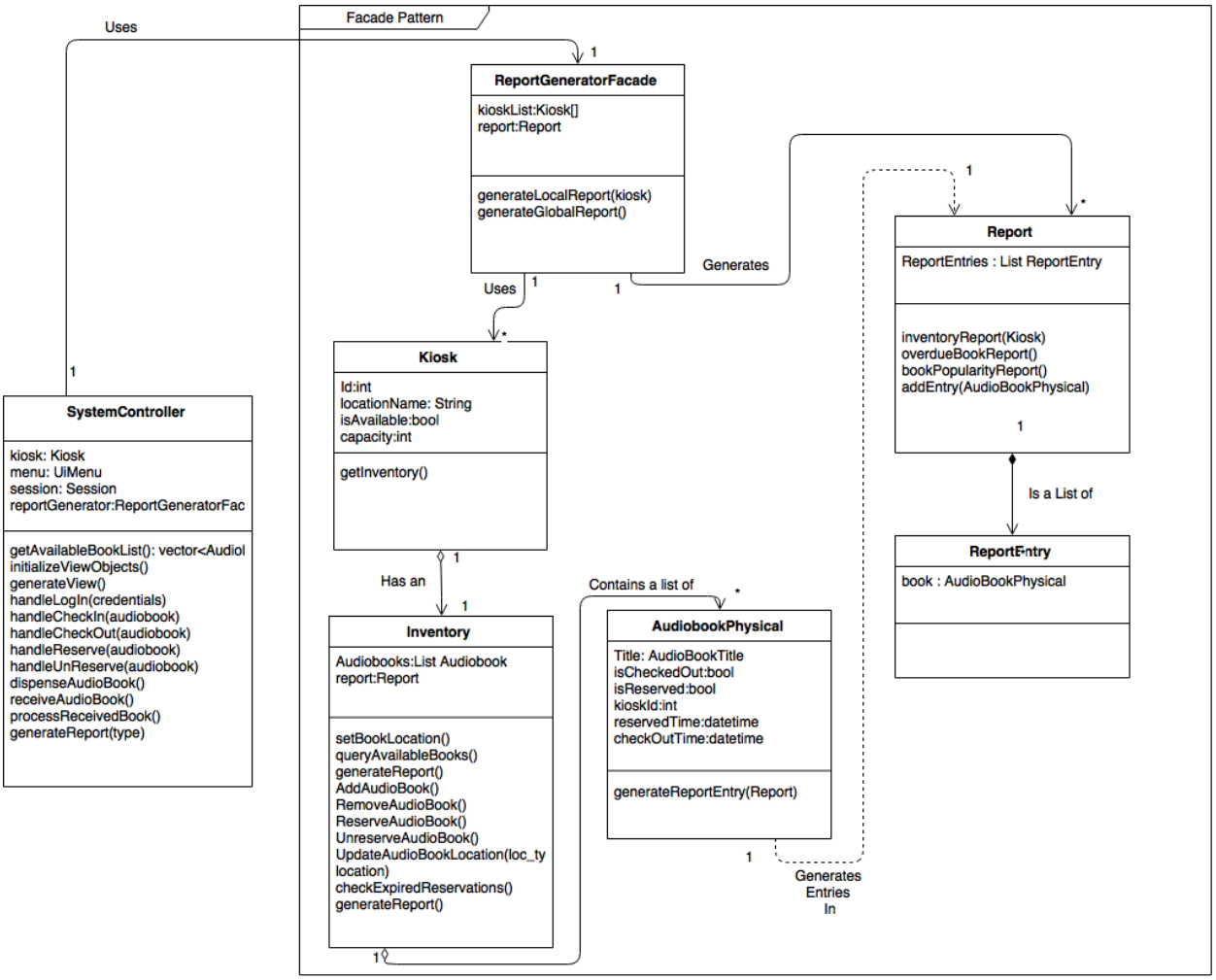
The BlueBox problem statement says that on-demand reports must be generated for managers. These reports include both the current inventory contents for one or more kiosks, and the statistics for each audio book in the system.

This action requires communication with the session object to validate the user's manager status, as well as one or more kiosk objects, and their associated inventory objects. This action also requires the creation of a new report object, and filling it with reportElement objects containing the lending statistics for each item in the inventory.

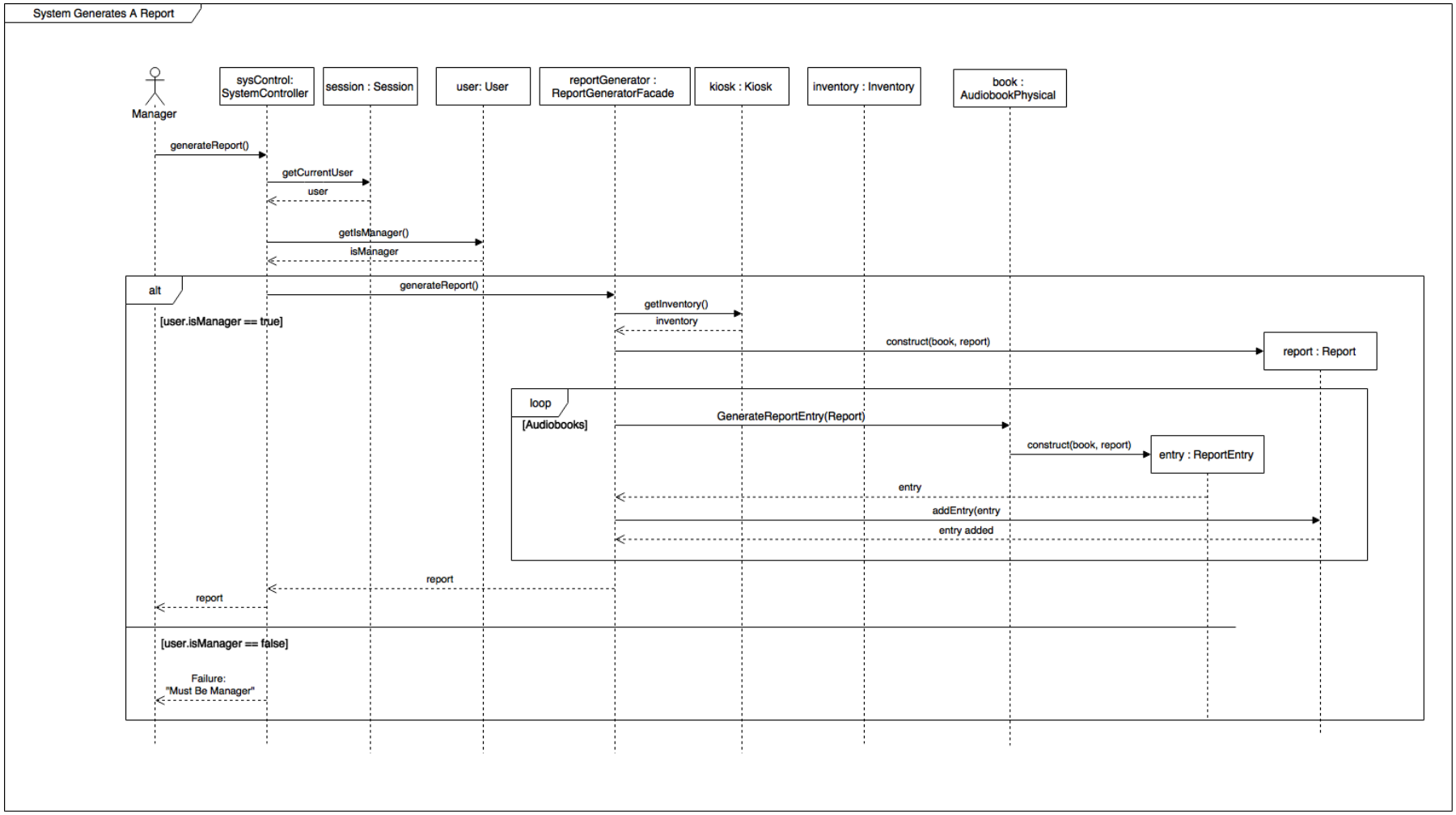
Rationale for façade pattern:

The coordination of at least 3 major different object classes seemed like a good fit for a façade pattern. The façade would introduce a very simplified interface, `Report &generateReport(Kiosk[] kiosks)`, and hide the complicated details of the communication with the inner objects.

Simplified portion of the class diagram showing the façade pattern in use:



Updated sequence diagram using new façade pattern:



Design Pattern 2: *Observer Pattern*

Design problem to be solved:

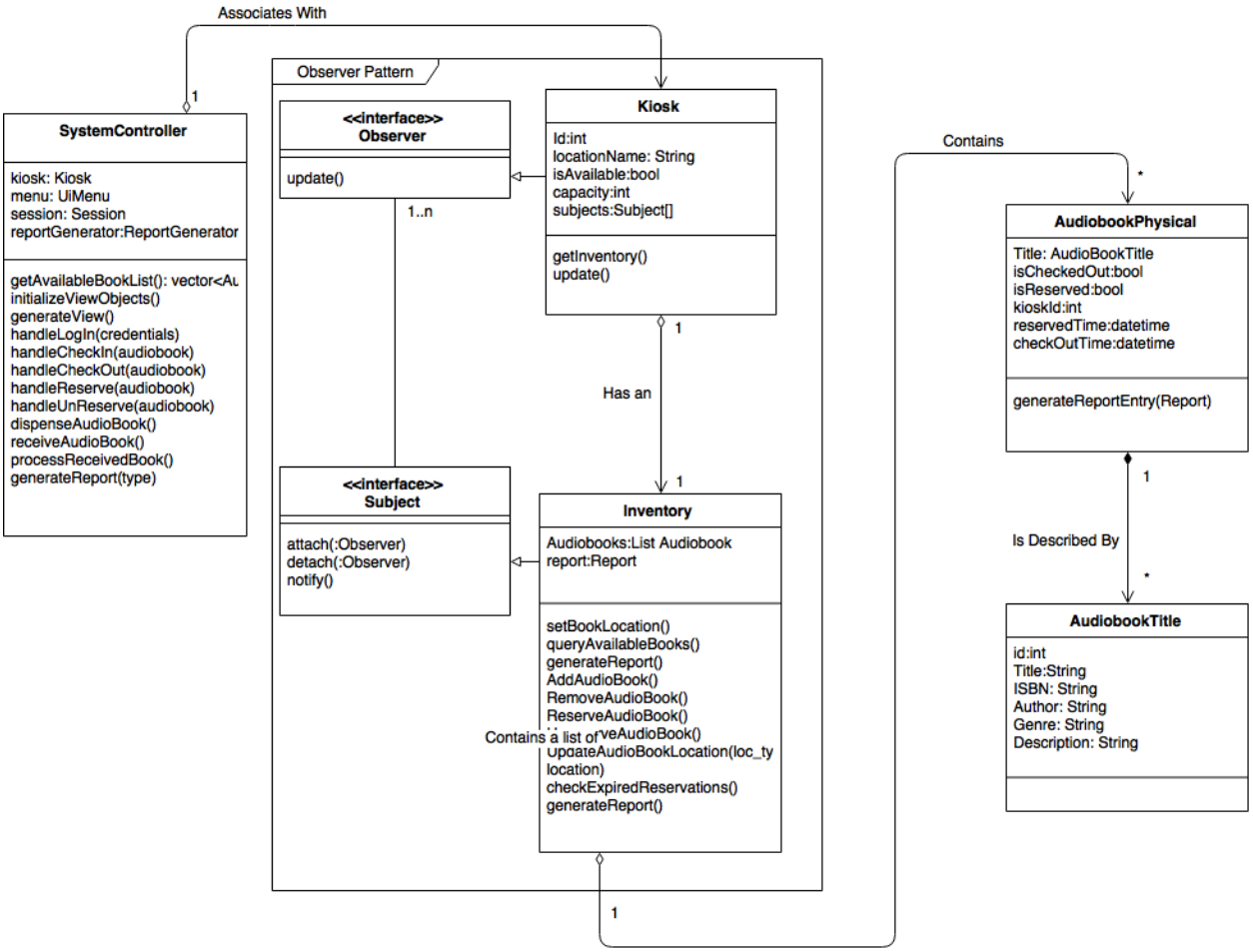
The BlueBox problem statement requires that are physical audiobooks are kept track of. This means that even if there are multiple events happening simultaneously or near-simultaneously, the inventory must be kept up-to-date as fast as possible.

For example, a manager could be generating a report via the on-line interface, while a user is attempting to reserve an audiobook via the on-line interface. And another user is checking out a book using the physical interface at a kiosk.

Rationale for observer pattern:

Keeping all of the associated objects (specifically all active session objects) up-to-date seems like a great fit for the observer pattern. Whenever an audiobook in the inventory is updated it can send a message to all of the active sessions (which are observers) so that they can either take action to update the user of the change or verify that the change does not affect that session.

Simplified portion of the class diagram showing the observer pattern in use:



Full updated class diagram reflecting all changes