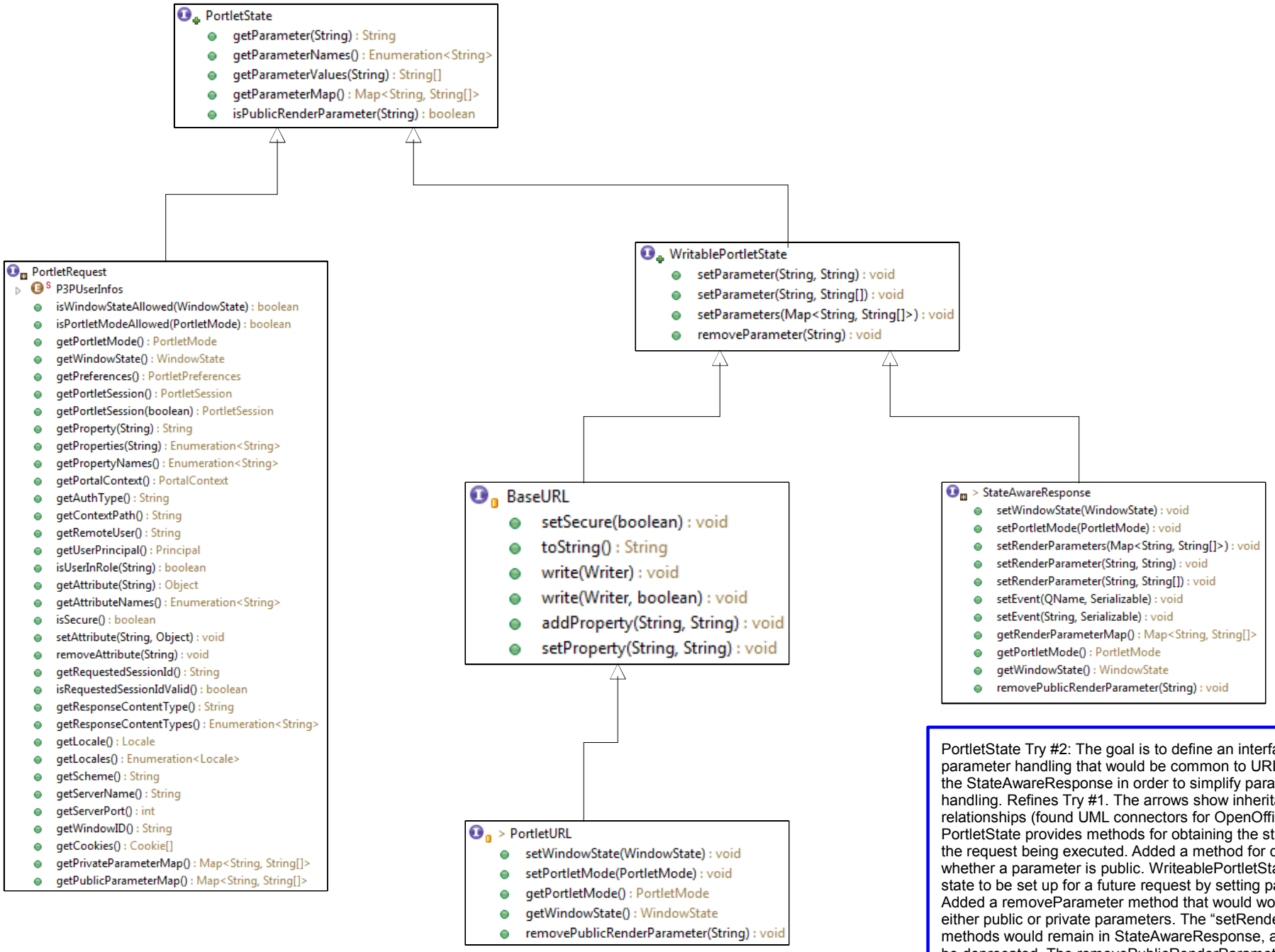
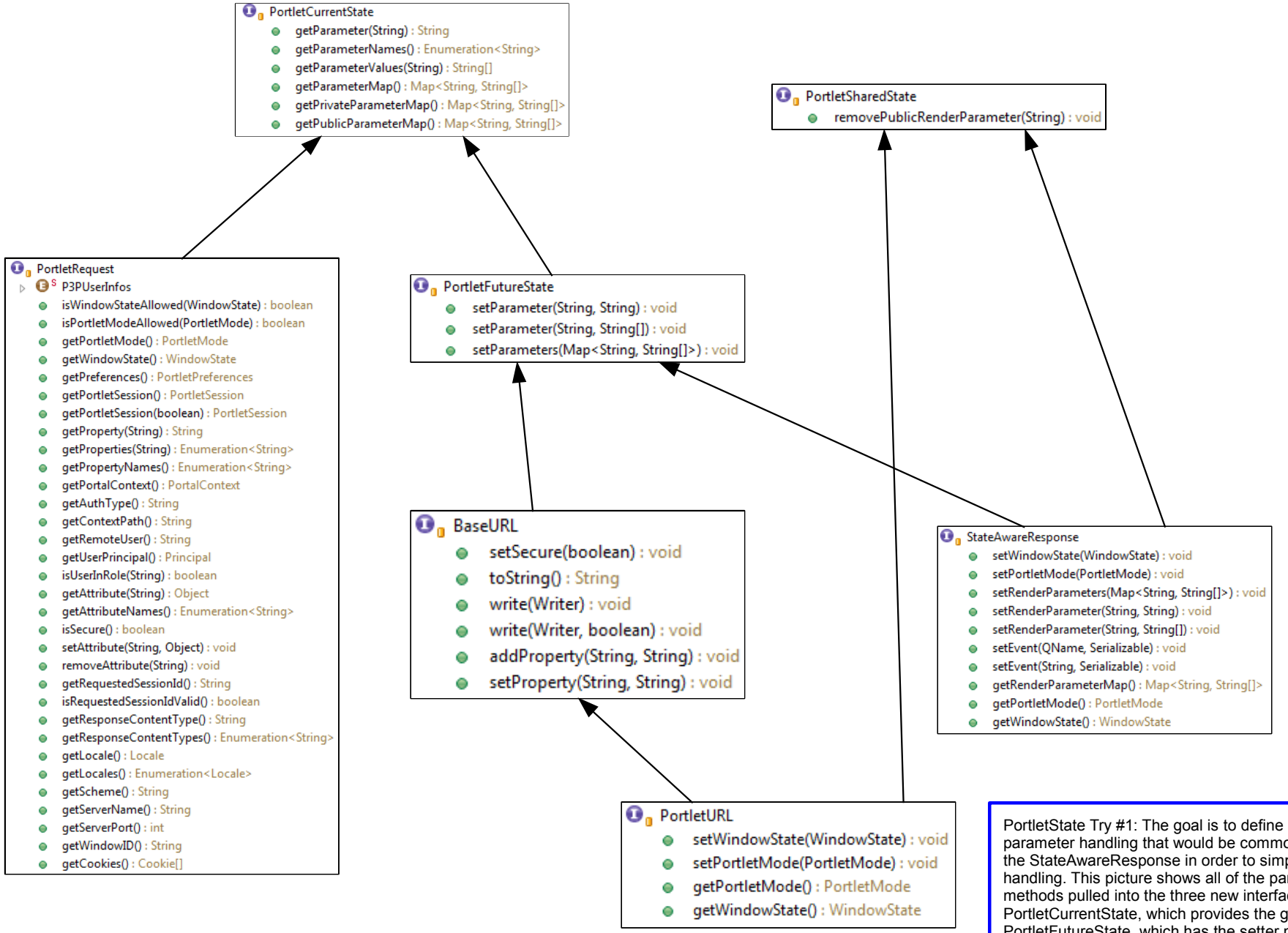


PortletState Try #2: The goal is to define an interface for parameter handling that would be common to URLs and to the StateAwareResponse in order to simplify parameter handling. Refines Try #1. The arrows show inheritance relationships (found UML connectors for OpenOffice, yay!) PortletState provides methods for obtaining the state valid for the request being executed. Added a method for determining whether a parameter is public. WriteablePortletState allows state to be set up for a future request by setting parameters. Added a removeParameter method that would work with either public or private parameters. The “setRender...” methods would remain in StateAwareResponse, and would be deprecated. The removePublicRenderParameter method would remain in PortletURL and StateAwareResponse and would be deprecated.
Question: Should PortletMode & WindowState be handled as “portlet state” in these interfaces? Or should we define an “ExtendedPortletState” containing the parameters, window state, and portlet mode?



PortletState Try #1: The goal is to define an interface for parameter handling that would be common to URLs and to the StateAwareResponse in order to simplify parameter handling. This picture shows all of the parameter handling methods pulled into the three new interfaces PortletCurrentState, which provides the getter methods, PortletFutureState, which has the setter methods, and PortletSharedState, which contains the single method that works only with public render parameters. The arrows show an inheritance relationship. PortletCurrentState allows you to get the parameters associated with the request currently being executed, while PortletFutureState allows you to set up state for a future request. The future request would be a render request if the state is set on the StateAwareResponse. If set on a URL, it would be the state that would come into play if the URL is activated.