Photo Album test specification

This is a FineFit specification of a photo album application. The application consists of a client that shows photos in a sequence and a server that stores the photos. We can change the content of the photos in the client but then must synchronize them with the server. Any new photos must be added to the server, and photos that exist in the server but we have asked to delete should be removed from the server. The following specification describes how each operation should behave.

Each photo is identified by a unique photo id. We will use the set PID to represent these identifiers.

|  |  |
| --- | --- |
| Atom | Scope |
| PID | 3 |

The state space of the application consists of four relations:

|  |  |
| --- | --- |
| Relation | Type |
| album | seq PID |
| toAdd | set PID |
| existing | set PID |
| toDelete | set PID |

The invariant determines when a system state is valid. For this application we demand that:

1. the album is an injective sequence, that is a photo can appear in at most one location in the sequence,
2. the photos we are about to send to the server do not already exist in the server,
3. the photos we are about to delete from the server are indeed in the server,
4. we do not ask to add a photo and delete it at the same time

|  |
| --- |
| Invariant |
| album in Int lone -> lone PID |
| no toAdd & existing |
| toDelete in existing |
| no toAdd & toDelete |
| #album <= 3 |

When the application begins there are no photos in the album and no photos in the server.

|  |  |
| --- | --- |
| sample.OperationFixture |  |
| init |  |
|  | some PID |
| album | { i:Int, p : PID | p != p } |
| toAdd | { p :PID | p ! =p } |
| toDelete | { p :PID | p ! =p } |
| existing | { p :PID | p ! =p } |

The operation addPhoto appends a photo to the album, provided that the photo is not already in the album and that the album has enough space:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| sample.OperationFixture |  |  |  |  |
| addPhoto |  |  |  |  |
| pid:PID | #album < 3 and  pid !in toAdd + existing | #album < 3 and pid in toDelete | pid in album.elems | #album = 3 |
| album | album.add[pid] | album.add[pid] | album | album |
| toAdd | toAdd + pid | toAdd | toAdd | toAdd |
| toDelete | toDelete | toDelete - pid | toDelete | toDelete |

The operation removePhoto takes an index into the album sequence and removes the photo from the album. All other photos shift to the right. If the photo is new (and therefore was about to be added to the server) we have to remove it from toAdd to ensure it won't be added to the server when we save the album:

|  |  |  |
| --- | --- | --- |
| sample.OperationFixture |  |  |
| removePhoto |  |  |
| i:Int | i in album.PID and album[i] in toAdd | i in album.PID and album[i] in existing |
| album | album.delete[i] | album.delete[i] |
| toAdd | toAdd - album[i] | toAdd |
| toDelete | toDelete | toDelete + album[i] |

The operation save sends all the newly added photos to the server and removes from the server all the old photos that we have asked to remove:

|  |  |
| --- | --- |
| sample.OperationFixture |  |
| save |  |
|  | some PID |
| toAdd | { p : PID | p != p } |
| toDelete | { p : PID | p != p } |
| existing | existing + toAdd - toDelete |