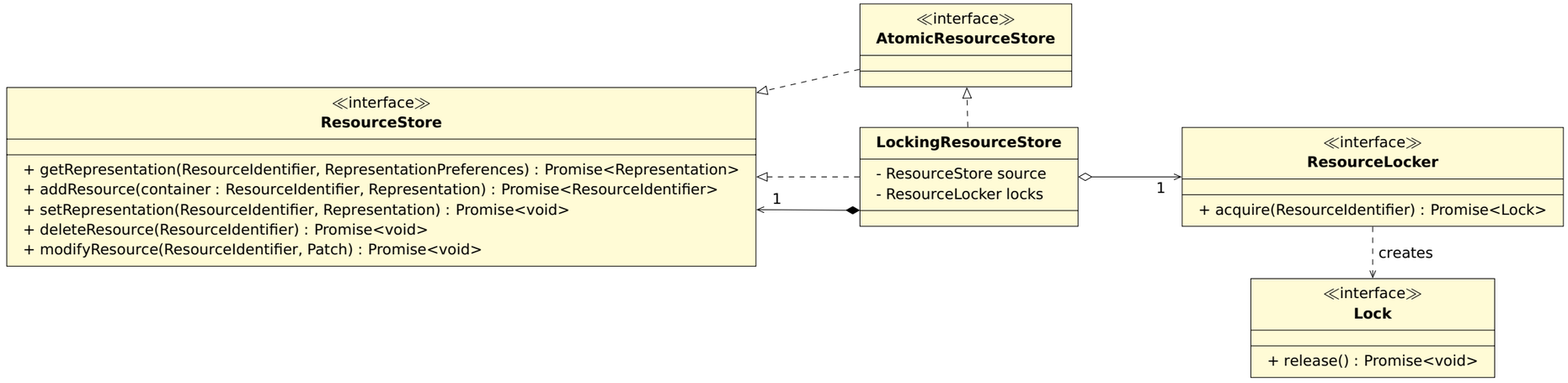
Solid server – Store atomicity *(status: draft)*

Ruben Verborgh – August 12, 2019

ResourceStore and atomic operations



The **ResourceStore** interface has been designed such that each of its methods can be implemented in an *atomic* way: for each CRUD operation,[1] only one dedicated method needs to be called. It is up to the implementer of the interface to (not) make an implementation atomic. For some implementations, such as triple stores or other database back-ends, atomicity is a given. We *could* explicitly indicate atomicity by having such implementations implement the (otherwise empty) **AtomicResourceStore** interface as a tag.

Some implementations are *not* atomic by default, such as a file system, where a read+append sequence could unknowingly be interrupted by a write that thereby breaks atomicity. Such non-atomic stores could be made atomic by decorating them with a **LockingResourceStore**. This class wraps another **ResourceStore** with a locking mechanism, which can be implemented in different ways. An example method implementation is listed on the right.

```
async function modifyResource(id, patch) {
  const lock = await this._locks.acquire(identifier);
  try { return await this._source.modifyResource(id, patch); }
  finally { await lock.release(); }
}
```

[1] There are 5 operations rather than 4 because we distinguish between full representations update for `PUT` and partial updates for `PATCH`.