SOEN 387 Web-based Enterprise Application Design

Stuart Thiel

Concordia University
Department of Computer & Software Engineering

Fall, 2015

Outline

Dependant Mapping

Unit of Work

Information of Limited Scope

- There is a primary object that we care about in general
- There is information that is only relevant with respect to that object
- The object is relevant without that information

Phone Numbers Example

- What if a household is identified by primary phone number?
- Does this makes sense still?
- Did it support some unseen bias?

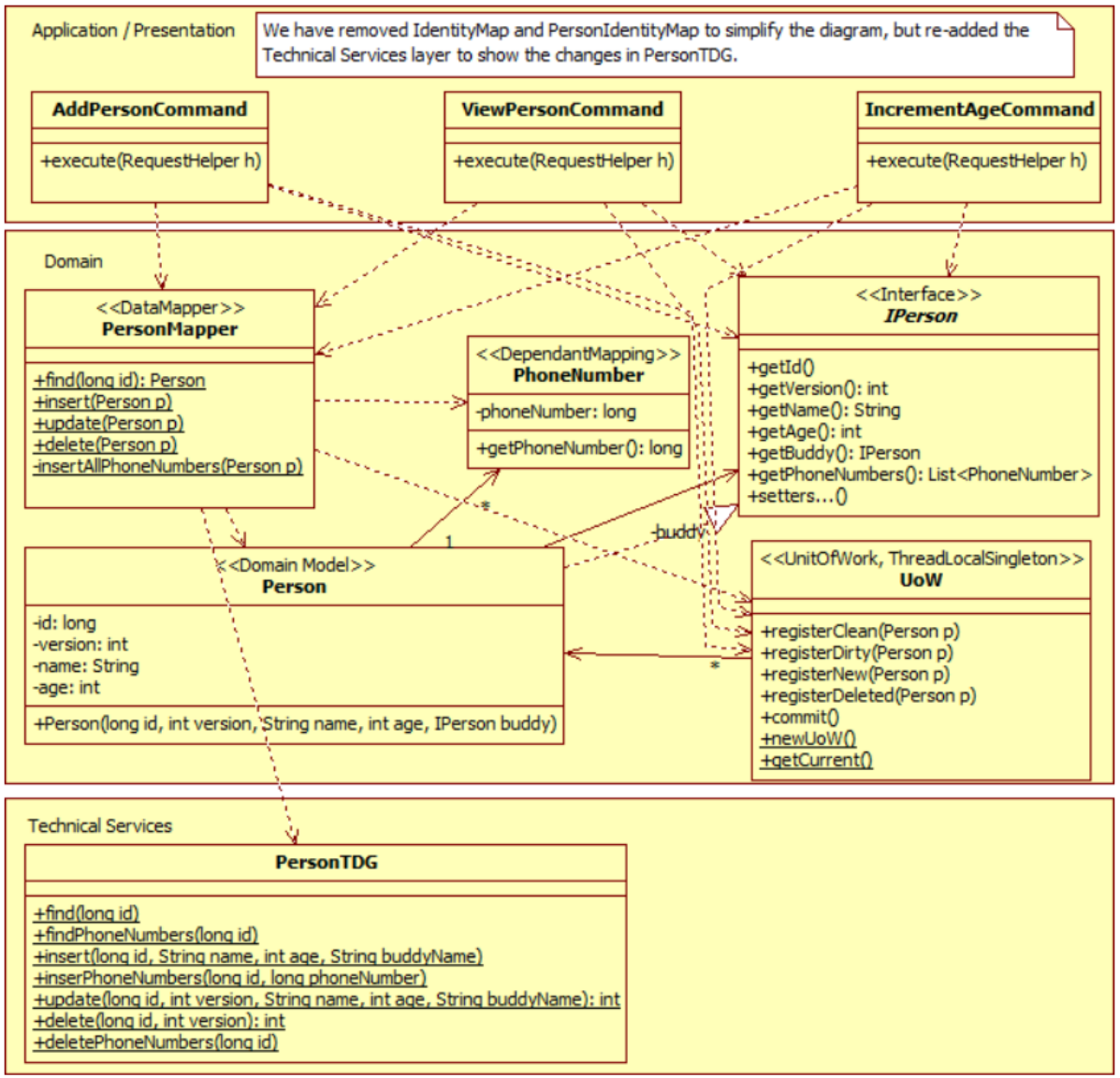
Phone Numbers Example Continued

- A person registers with all their phone numbers
- What happens when we delete that person?
- Note that such relations are often many-to-one
- Does a Phone Number deserve its own object?
- How do we update it?

Dependant Mapping

- Dependant Mappings depend on id and version of the instance that references them
- They may not even need their own objects if they are primitive types
- They are kept in a separate table with a key for the parent object
- The mapper for the parent object maps the dependant objects.
- The TDG for the parent object also likely has the SQL to interact with the dependant objects.
- Modifying dependant objects can mean updating the version of the parent object.

Dependant Mapping Class Diagram



Dependant Mapping Code 1

```
public static Person find(long id) throws Exception {
        if (PersonIdentityMap.has(id))
                return PersonIdentityMap.get(id);
        ResultSet rs = PersonTDG.find(id);
        if (rs.next()) {
                List<PhoneNumber> numbers = new Vector<PhoneNumber>();
                Person p = new Person(id, rs.getInt("p.version"),
                                rs.getString("p.name"), rs.getInt("p.age"),
                                new PersonProxy(rs.getLong("p.buddy")), numbers);
                rs.close();
                rs = PersonTDG.findPhoneNumbers(id);
                while (rs.next()) {
                        numbers.add(new PhoneNumber(rs.getLong("pn.number")));
                }
                rs.close();
                PersonIdentityMap.put(id, p);
                UoW.getCurrent().registerClean(p);
                return p;
        }
        return null;
}
```

Dependant Mapping Code 2

```
public static void insert(Person p) {
        PersonTDG.insert(/* ... */);
        insertAllPhoneNumbers(p);
}

public static void update(Person p) {
        PersonTDG.update(/* ... */);
        PersonTDG.deletePhoneNumbers(p.getId());
        insertAllPhoneNumbers(p);
}

public static void delete(Person p) {
        PersonTDG.delete(/* ... */);
        PersonTDG.deletePhoneNumbers(p.getId());
}

private static void insertAllPhoneNumbers(Person p) {
        for (PhoneNumber phoneNumber : p.getPhoneNumbers()) {
                PersonTDG.insertPhoneNumber(p.getId(),
                  phoneNumber.getPhoneNumber());
        }
}
```

How to Manipulate Many Domain Objects?

- What if a Use Case is complex, many things change?
- Some added, some deleted, some updated?
- We need to keep all changes to DB consistent
- We need to be efficient in our DB requests

Unit of Work

- A Pure Fabrication to track in-memory objects during a transaction
- Tracks State
- Ensures Referential integrity
- Commits changes
- One Unit of Work for ALL objects

Keep Track of In-Memory State

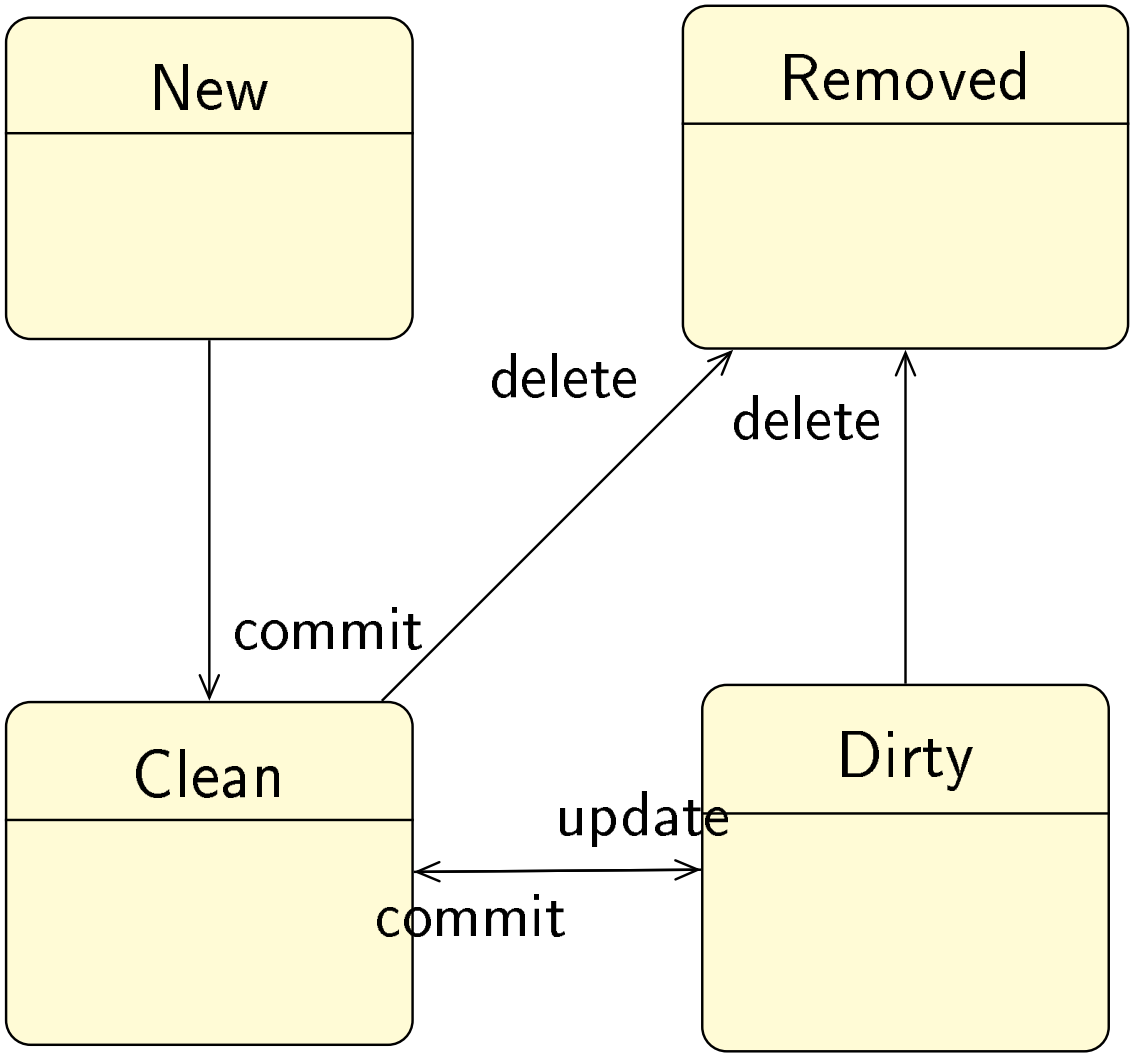
- Clean: same as what is in DB
- New: not in the DB yet
- Removed: should be removed from DB
- Dirty: In-memory is more up-to-date than DB

Referential Integrity

- Some of you have asked about referential integrity
- In tracking object state, we have an opportunity to enforce this
- UoW should take care of this

State Lifecycle

These are the transitions that make sense. . . what about others?



UoW with Identity Map

- Some of you may have noticed some code from PersonMapper's find method
- What does Identity Map need to function as an Identity Map?

```
PersonIdentityMap.put(id, p);
```

Identity Map as Interface on UoW

- If UoW tracks clean objects this is an easy solution!

Avoids Deadlocks

- Database-level integrity constraints can be tricky
- UoW can know necessary order so programmer can Add/Remove/Update at will
- UoW will make sure all is done in proper order
- Doing all database changes at once is also faster/less prone to lost updates

UoW and Context

- Since Identity Maps are per-request, should UoW be per-request?
- Fowler says not always, but thar lies danger. . . LOTS of shared memory
- Sharing a UoW is worse since it also involves tracking changes
- Could there be a reason to use two UoWs?
- If Logging/tracing done separately?
- HOWEVER! If you do need to collect changes over many requests, theoretically this could work
- Avoid if possible (it almost always is)