This page last changed on Jul 09, 2010 by patrik_nordwall.

Sculptor Port of **DDD Sample**

This page describes a port of the DDD Sample using Sculptor. It is a domain model based on the cargo example used in Eric Evans' book.

Sculptor will help you with the static structure of the application. It can generate the data part of the domain objects and boilerplate code for services and repositories. It has a nice runtime framework for especially the persistence implementations, such as standard queries. However, to qualify as a true DDD application the domain objects must have complex behavior - business logic. With this sample we illustrate how to integrate manually written logic with the automatically generated parts.

Table of Contents:

Model

The definition of the domain model, using Sculptor's textual DSL is separated in one file for each Module, plus one "empty" top file, which imports the other files.

```
import "platform:/resource/DDDSample/src/main/resources/model-location.btdesign"
import "platform:/resource/DDDSample/src/main/resources/model-cargo.btdesign"
import "platform:/resource/DDDSample/src/main/resources/model-carrier.btdesign"
import "platform:/resource/DDDSample/src/main/resources/model-routing.btdesign"

Application DDDSample {
    basePackage=org.sculptor.dddsample

}
```

```
import "platform:/resource/DDDSample/src/main/resources/model-location.btdesign"
import "platform:/resource/DDDSample/src/main/resources/model-carrier.btdesign"
import "platform:/resource/DDDSample/src/main/resources/model-routing.btdesign"

ApplicationPart CargoPart {

  Module cargo {
      "A Cargo. This is the central class in the domain model,
      and it is the root of the Cargo-Itinerary-Leg-DeliveryHistory aggregate.

      A cargo is identified by a unique tracking id, and it always has an origin
      and a destination. The life cycle of a cargo begins with the booking procedure,
      when the tracking id is assigned. During a (short) period of time, between booking
      and initial routing, the cargo has no itinerary.

      The booking clerk requests a list of possible routes, matching a route specification,
      and assigns the cargo to one route. An itinerary listing the legs of the route
      is attached to the cargo.
```

A cargo can be re-routed during transport, on demand of the customer, in which case the destination is changed and a new route is requested. The old itinerary, being a value object, is discarded and a new one is attached.

It may also happen that a cargo is accidentally misrouted, which should notify the proper personnel and also trigger a re-routing procedure.

The life cycle of a cargo ends when the cargo is claimed by the customer.

The cargo aggregate, and the entre domain model, is built to solve the problem of booking and tracking cargo. All important business rules for determining whether or not a cargo is misrouted, what the current status of the cargo is (on board carrier, in port etc), are captured in this aggregate."

```
Entity Cargo {
    - @TrackingId trackingId key
    - @Location origin required fetch="join"
    - @Location destination required fetch="join"
    - @Itinerary itinerary nullable fetch="join" cascade="all" inverse opposite cargo
    - Set<@HandlingEvent> events cascade="none" opposite cargo

    Repository CargoRepository {
        @Cargo find(@TrackingId trackingId) throws CargoNotFoundException => FindCargoAccessObject;
        @Cargo find(@TrackingId trackingId, boolean loadDeliveryHistory) throws
CargoNotFoundException;
        protected populateAssociations;
        findAll;
        save;
        TrackingId nextTrackingId;
        detachItineray(Cargo cargo);
        "Delete orphaned itineraries - conceptually the responsibility
        of the Cargo aggregate"
        protected deleteOrphanItinerary => AccessObject;
        protected findById;
    }
}

"Uniquely identifies a particular cargo. Automatically generated by the application."
BasicType TrackingId {
    String identifier key;
}

ValueObject Itinerary {
    not immutable
    not optimisticLocking
    belongsTo @Cargo
    - @Cargo cargo opposite itinerary
    - List<@Leg> legs inverse
}

"An itinerary consists of one or more legs."
ValueObject Leg {
    belongsTo @Cargo
    - @CarrierMovement carrierMovement fetch="join"
    - @Location from databaseColumn="FRM" fetch="join"
    - @Location ^to databaseColumn="T" fetch="join"
}

ValueObject DeliveryHistory {
    not persistent
    - Set<@HandlingEvent> events
}

"Cargo booking service."
Service BookingService {
    inject @CargoRepository
```

```
    "Registers a new cargo in the tracking system, not yet routed."
    @TrackingId bookNewCargo(@UnLocode origin, @UnLocode destination)
       throws org.sculptor.dddsample.location.exception.LocationNotFoundException;

    "Requests a list of itineraries describing possible routes for this cargo."
    List<@Itinerary> requestPossibleRoutesForCargo(@TrackingId trackingId)
       throws CargoNotFoundException,
          org.sculptor.dddsample.location.exception.LocationNotFoundException;

    "Assigns a cargo to route."
    void assignCargoToRoute(@TrackingId trackingId, @Itinerary itinerary) throws
CargoNotFoundException;

    protected findLocation => LocationService.find;
    protected fetchRoutes => RoutingService.fetchRoutesForSpecification;
}

ValueObject RouteSpecification {
    not persistent
    - @Location origin
    - @Location destination
    DateTime arrivalDeadline
}

"Represents the different status codes for a cargo."
enum StatusCode {
    NOT_RECEIVED, IN_PORT, ONBOARD_CARRIER, CLAIMED, UNKNOWN
}

Service RoutingService {
    "A list of itineraries that satisfy the specification. May be an empty list if no route is found."
    List<@Itinerary> fetchRoutesForSpecification(@RouteSpecification routeSpecification)
       throws org.sculptor.dddsample.location.exception.LocationNotFoundException;
    protected findShortestPath => GraphTraversalService.findShortestPath;
    protected findLocation => LocationService.find;
    protected findCarrierMovement => CarrierService.find;
    protected saveCarrierMovement => CarrierService.save;
}

Service TrackingService {
    inject @CargoRepository

    "Track a particular cargo."
    @Cargo track(@TrackingId trackingId) throws CargoNotFoundException;

    "Inspect cargo and send relevant notifications to interested parties,
    for example if a cargo has been misdirected, or unloaded
    at the final destination."
    void inspectCargo(@TrackingId trackingId) throws CargoNotFoundException;
}

ValueObject HandlingEvent {
    - @Type type
    - @CarrierMovement carrierMovement nullable fetch="join"
    - @Location location  fetch="join"
    DateTime completionTime
    DateTime registrationTime
    - @Cargo cargo  fetch="join" opposite events

    Repository HandlingEventRepository {
       save;
       "All handling events for this cargo, ordered by completion time."
       List<@HandlingEvent> findEventsForCargo(@TrackingId trackingId);
       protected findByQuery;
    }
}
```

```
    enum Type {
        boolean carrierMovementRequired

        LOAD("true"),
        UNLOAD("true"),
        RECEIVE("false"),
        CLAIM("false"),
        CUSTOMS("false")
    }

    Service HandlingEventService {
        inject @CargoRepository
        inject @HandlingEventRepository

        void register(DateTime completionTime, @TrackingId trackingId,
            @CarrierMovementId carrierMovementId, @UnLocode unlocode, @Type type)
          throws CargoNotFoundException,
              org.sculptor.dddsample.carrier.exception.CarrierMovementNotFoundException,
              org.sculptor.dddsample.location.exception.LocationNotFoundException;
        protected findCarrierMovement => CarrierService.find;
        protected findLocation => LocationService.find;
    }

  }

}
```

```
ApplicationPart LocationPart {

    Module location {
        "A location is our model is stops on a journey, such as cargo
        origin or destination, or carrier movement endpoints."
        Entity Location {
            not optimisticLocking
            - @UnLocode unLocode key
            'Actual name of this location, e.g. "Stockholm"'
            String name not changeable

            Repository LocationRepository {
                @Location find(@UnLocode unLocode) throws LocationNotFoundException;
                findAll;
                protected findByKeys;
            }
        }

        "United nations location code."
        BasicType UnLocode {
            String unlocode key
        }

        Service LocationService {
            find => LocationRepository.find;
        }
    }

}
```

```
  import "platform:/resource/DDDSample/src/main/resources/model-location.btdesign"
```

```
ApplicationPart CarrierPart {

    Module carrier {
        "A carrier movement is a vessel voyage from one location to another."
        Entity CarrierMovement {
            not optimisticLocking
            not auditable
            - @CarrierMovementId carrierMovementId key databaseColumn="CARRIERMOVEMENT"
            - @Location from not changeable databaseColumn="FRM" fetch="join"
            - @Location ^to not changeable databaseColumn="T" fetch="join"

            Repository CarrierMovementRepository {
                @CarrierMovement find(@CarrierMovementId carrierMovementId) throws
 CarrierMovementNotFoundException;
                protected findByKeys;
                save;
            }
        }

        BasicType CarrierMovementId {
            String identifier key;
        }

        Service CarrierService {
            find => CarrierMovementRepository.find;
            save => CarrierMovementRepository.save;
        }
    }


}
```

```
ApplicationPart RoutingPart {

    Module routing {
        ValueObject TransitEdge {
            String carrierMovementId
            String fromUnLocode
            String toUnLocode
        }

        ValueObject TransitPath {
            - List<TransitEdge> transitEdges inverse
        }

        Service GraphTraversalService {
            inject @RtCarrierMovementRepository
            inject @RtLocationRepository
            List<@TransitPath> findShortestPath(String originUnLocode, String destinationUnLocode);
        }

        ValueObject RtCarrierMovement {
            String carrierMovementId key
            - @RtLocation from databaseColumn="FRM" fetch="join"
            - @RtLocation ^to databaseColumn="T" fetch="join"

            Repository RtCarrierMovementRepository {
                inject @RtLocationRepository
                storeCarrierMovementId(String cmId, String from, String to);
                protected save;
            }
        }
```

```
        ValueObject RtLocation {
            String unlocode key

            Repository RtLocationRepository {
                List<String> listLocations();
                protected findAll;
                findByKeys;
            }
        }
    }

  }
```
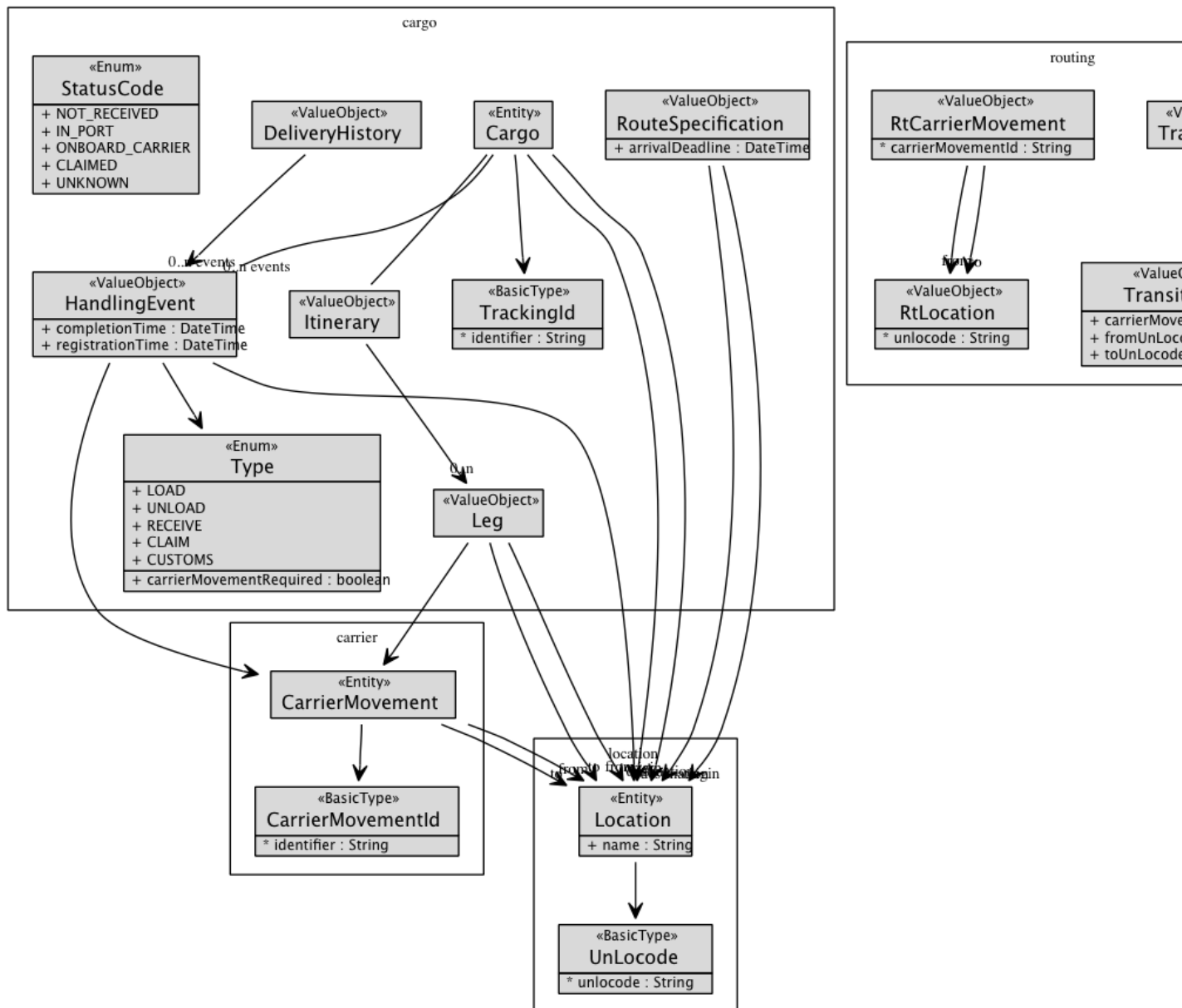
The generated vizualization with [graphviz](#) looks like this:

JUnit Tests

This sample has an extensive test suite, which illustrates how to write junit tests at different levels.

- logic in the domain objects is tested with ordinary junit tests, without any need for database emulation or spring container
- repositories and details of the persistence are tested with DBUnit tests and spring container and hibernate using in memory database
- services are tested in a similar way as the repositories, but also using easymock framework

Metrics

It is interesting to compare the fully hand written original DDD Sample with this partly generated port. The number of hand written lines of code in the Sculptor port is 783. Compared to 1179 in the original. The business logic is almost identical, but the Sculptor variant has less lines of code since much of the boring boilerplate code is generated. It has bean measured with JavaNCSS.
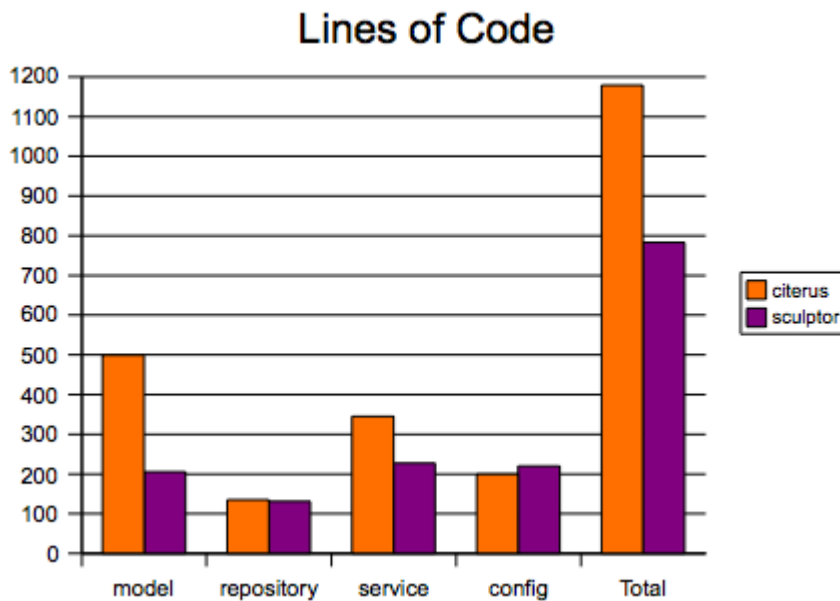
org.sculptor.dddsample.cargo.accessimpl	18		
org.sculptor.dddsample.cargo.domain	172		
org.sculptor.dddsample.cargo.repositoryimpl	44		
org.sculptor.dddsample.cargo.serviceimpl	165		
org.sculptor.dddsample.carrier.domain	0		
org.sculptor.dddsample.carrier.repositoryimpl	18		
org.sculptor.dddsample.carrier.serviceimpl	7		
org.sculptor.dddsample.common	3		
org.sculptor.dddsample.location.domain	22		
org.sculptor.dddsample.location.repositoryimpl	18		
org.sculptor.dddsample.location.serviceimpl	7		
org.sculptor.dddsample.routing.domain	8		
org.sculptor.dddsample.routing.repositoryimpl	33		
org.sculptor.dddsample.routing.serviceimpl	48		
model.design	220		
hbm	0		
spring	0		
se.citerus.dddsample.application.persistence	65		
se.citerus.dddsample.application.remoting	75		
se.citerus.dddsample.application.routing	42		
se.citerus.dddsample.domain.model	12		
se.citerus.dddsample.domain.model.cargo	295		
se.citerus.dddsample.domain.model.carrier	56		
se.citerus.dddsample.domain.model.handling	72		
se.citerus.dddsample.domain.model.location	64		
se.citerus.dddsample.domain.service	57		
se.citerus.dddsample.domain.service.impl	141		
se.citerus.routingteam	30		
se.citerus.routingteam.internal	70		
hbm	90		
spring	110		

	citerus	sculptor
model	499	205
repository	135	131
service	345	227
config	200	220
Total	1179	783

JUnit tests are not included, since they are very similar. Web application is not included, since it is not implemented in this Sculptor port.

Try It

If you are only interested in a sneak preview of the source code you can download it here: Sculptor-DDDSample-src.zip

Another choice is to try the real thing:

1. Install Sculptor and its requisites as described in the Installation Guide.

2. Checkout the source code from Subversion: https://fornax.svn.sourceforge.net/svnroot/fornax/trunk/cartridges/sculptor/DDDSample

3. Build with `mvn clean install`. This will also run all JUnit tests located in `src/test/java`. Take a look at some of them and run them from Eclipse also.

4. Study the `.btdesign` files located in `src/main/resources`.

5. Study the hand written code in `src/main/java` and the generated code in `src/generated/java`.

Learn more about the capabilities of Sculptor by reading the Hello World and Advanced Tutorial

Source

The complete source code for this sample is available in Subversion.

Web Access (read only):
http://fisheye3.cenqua.com/browse/fornax/trunk/cartridges/sculptor/DDDSample

Anonymous Access (read only):
https://fornax.svn.sourceforge.net/svnroot/fornax/trunk/cartridges/sculptor/DDDSample