Sample Application

Axon Framework 1.2

Jettro Coenradie

Allard Buijze

Table of Contents

1. Address Book

1.1. Introduction

Every good framework needs a sample application. You want to have an application that demonstrates enough of the framework and is easy enough to understand. The Address book sample we have created is such an application. A very simple domain to understand with enough features to show things like the event mechanism, the commands and the separate data store and query database.

The address book sample shows an Adobe flex user interface. The flex client makes the event handling interesting. The server side domain events are pushed to the flex clients to keep them up to date.

So what does the sample provide? The sample contains contacts with addresses. Each contact can have a maximum of three addresses, one for each available type. Using the query database, you can request all contacts, request contact details and search the addresses. For this kind of queries, the query database is used. You can also create new contacts and addresses or provided updates.

To interact with the application, you have to dispatch comamnds. The domain creates events for things like: contact created, contact updated, address changed. These events are picket up by special listeners that publish these events to all flex clients using BlazeDS. More on this part follows in section Section 1.2, “Flex client”.

The following image gives an overview of the architecture as discussed in this section.

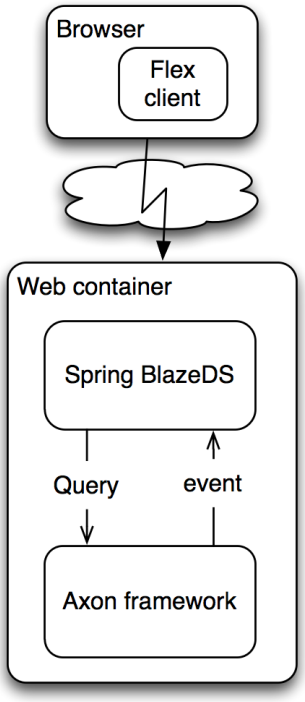

Figure 1.1. Architecture overview address book sample

The next image gives a preview of the flex client.

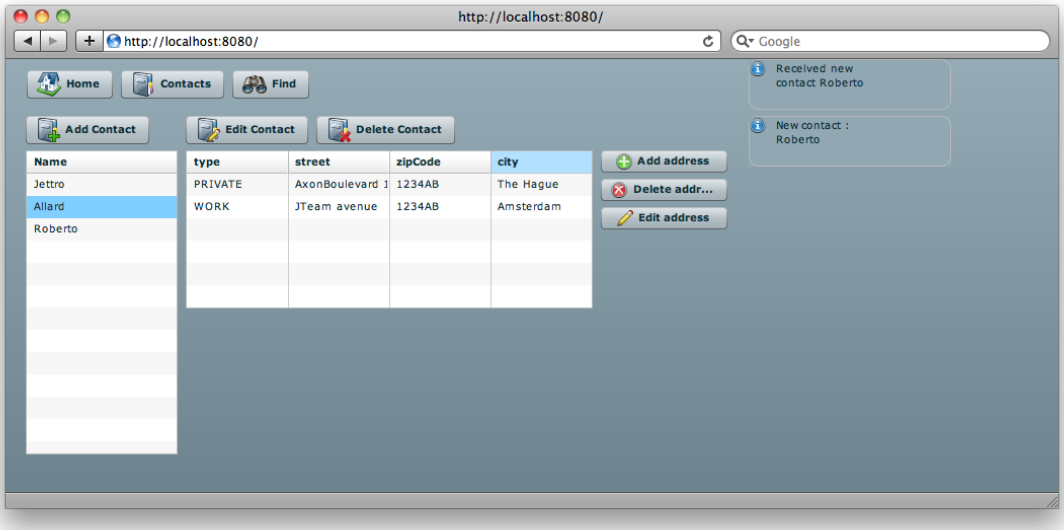

Figure 1.2. Preview of the flex client of the address book

1.2. Flex client

In this section we will describe the main components of the flex client. The flex client makes use of an Inversion Of Control framework called parsley. This framework makes the application maintainable and easier to understand. Parsley makes use of meta-data tags to indicate injection of objects and handling of messages. It makes heavily use of messages and events.

Note

For the flex application we use the parsley framework. More information about the framework can be found on the Parsley website: `http://www.spicefactory.org/parsley/`

The flex client talks to the server. Remote objects are used to ask data from the server and send data to the server. Another mechanism, called consumers, is used to receive update events. Both types of communication are handled by the BlazeDS framework. To make integration with the back-end even easier, the sample makes use of the spring blazeds integration project.

The nice part about using flex on the client side is that flex enables you to receive the domain events and keep a local cache of the data. In the end, the flex client becomes a remote cache of the data on the server that makes use of the domain events to stay up to date. To receive the events, the application registers an event listener. The following code block shows the event listener .

```
@Component
public class AddressListener {
    private final static Logger logger = LoggerFactory.getLogger(AddressListener.class);
    private UpdateMessageProducerForFlex producer;

    @EventHandler
    public void handleAddressCreatedEvent(AddressRegisteredEvent event) {
        AddressDTO addressDTO = AddressDTO.createFrom(
            event.getAddress(), event.getContactIdentifier(), event.getType());
        producer.sendAddressUpdate(addressDTO);
    }

    @EventHandler
    public void handleAddressRemovedEvent(AddressRemovedEvent event) {
        RemovedDTO removedDTO = RemovedDTO.createRemovedFrom(
            event.getContactIdentifier().toString(), event.getType());
        producer.sendRemovedUpdate(removedDTO);
    }

    @Autowired
    public void setProducer(UpdateMessageProducerForFlex producer) {
        this.producer = producer;
    }
}
```

This class registers two event listeners. The Axon Framework uses the EventHandler annotation and the argument of the method to register the right listener and to call this method when the event takes place. Both method create a DTO object and send this via the BlazeDS producer to all flex clients.

Note

Since this is not a reference manual for flex or parsley you will not find a detailed description of the flex client. If you want more information on the flex client, refer to this blog item. http://www.gridshore.nl/2010/02/25/creating-a-sample-for-axon-using-flex-and-parsley/

1.3. Application logic

Axon is a framework. Axon makes it easier to create your own application following the Axon principles. This section discusses the components of the sample that make use of the Axon Framework and that form the back-end of your application.

It all starts with the domain. The address book has a very easy domain, a root aggregate called Contact. The contact has a collection of addresses. The Address is an immutable value object. The most important business rule is that each address is of a certain type and a contact can only have one address of each type.

Providing new or updated data to the application is done using commands and command handlers. The `CreateContactCommand` is used to create new contacts, the `UpdateContactCommand` to update existing and more commands to provide data for other tasks. All these commands are handled by the `ContactCommandHandler`. The handler creates or updates the aggregate roots, in this case the`Contact`, and registers them with the repository. The aggregate root creates events like `ContactCreatedEvent`. These events are dispatched by the `ContactRepository` when storing the change has taken place. The following image gives you an idea what happens when a contact is created.

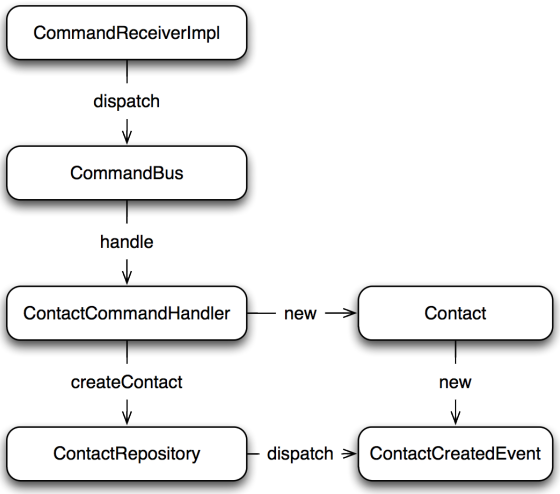

Figure 1.3. Interaction between the different components

As you can see from the previous image, events are dispatched by the repository. The listener as discussed in section Section 1.2, “Flex client” listens for these events.

Now that we have covered adding data by dispatching commands that are handled command handler class, the next part is about querying the data. The address book sample makes use of a separate data source

for querying the data. Two databases configurations are available, mysql and hsqldb. The configuration is available in the file database-context.xml. By changing the following lines you can switch between an hsqldb or mysql.

```
<bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
              <property name="locations" value="classpath:hsqldb.database.properties"/>
              </bean>
```

The sample makes use of jpa to keep the query datastore up te date. Events are received and handled by the`AddressTableUpdater`. The classes `ContactEntry` and `AddressEntry` represent the two entities as they are stored in the query datastore. The `ContactEntry` can be used to obtain a list of all available contacts. The `AddressEntry` can be used to query for addresses belonging to a contact and for addresses in a certain city.

It is important to understand that this query datasource registers listeners, just like the flex client, to get updates about contact and address changes.