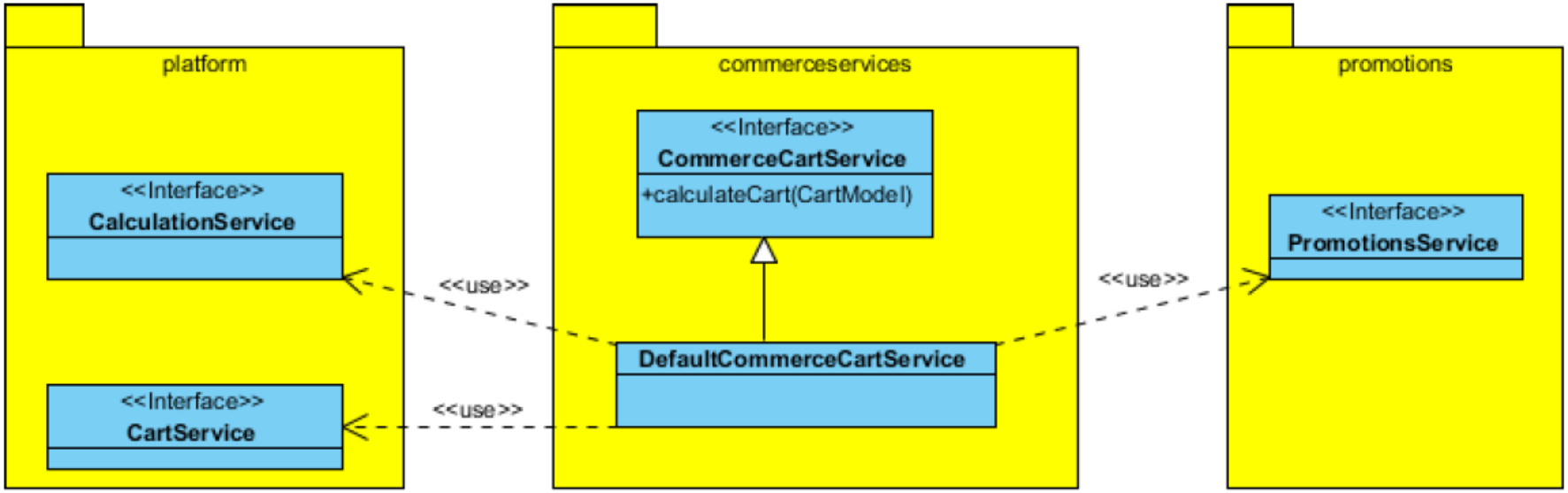
Commerceservices and Commercefecades

MODULE - 03

commerceservices

- Orchestrates platform and other extensions services to provide complete B2C use cases
- Extends more generic functionality from certain hybris extensions to add more B2C features



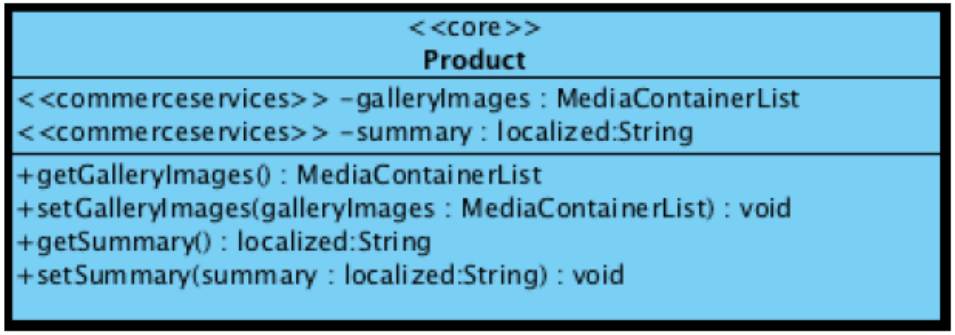
Data model: Product

summary

→ more concise product description (e.g. in search)

galleryImages

→ storing multiple images each resized to a number of standard formats expected by the storefront



commercefacades extension

Typical suite of storefront actions that make up a unified multichannel storefront API

- Viewing product details
- Adding a product to a cart
- Adding a delivery address during checkout
- Posting a review
- Searching for products with a free text search

Data Objects

- Facades return Data Objects to the Caller (Spring MVC Controller)
- Typically populated using a subset of data from the hybris ServiceLayer models
- Declared within `beans.xml`

commercefacades-beans.xml

```
<bean class="de.hybris.platform.commercefacades.order.data.DeliveryModeData">
    <property name="code" type="String"/>
    <property name="name" type="String"/>
    <property name="description" type="String"/>
    <property name="deliveryCost"
        type="de.hybris.platform.commercefacades.product.data.PriceData"/>
</bean>
```

Converters and Populators

Converter

- Interface for a converter that transforms an object of type A into an object of type B

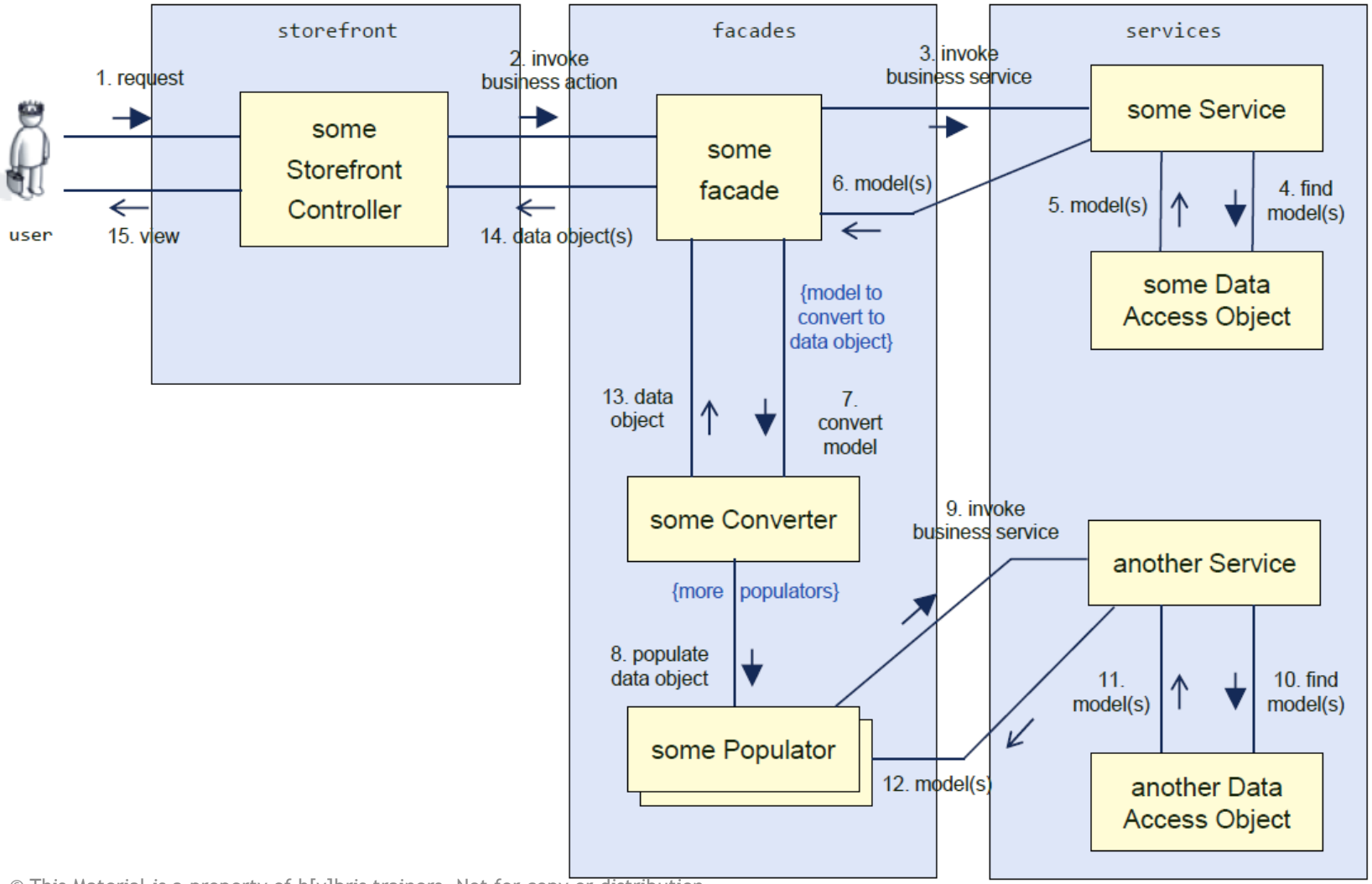
Populator

- Interface for a populator that sets values in a target instance based on values in the source instance
- Type conversion is typically broken down into a pipeline of Population steps

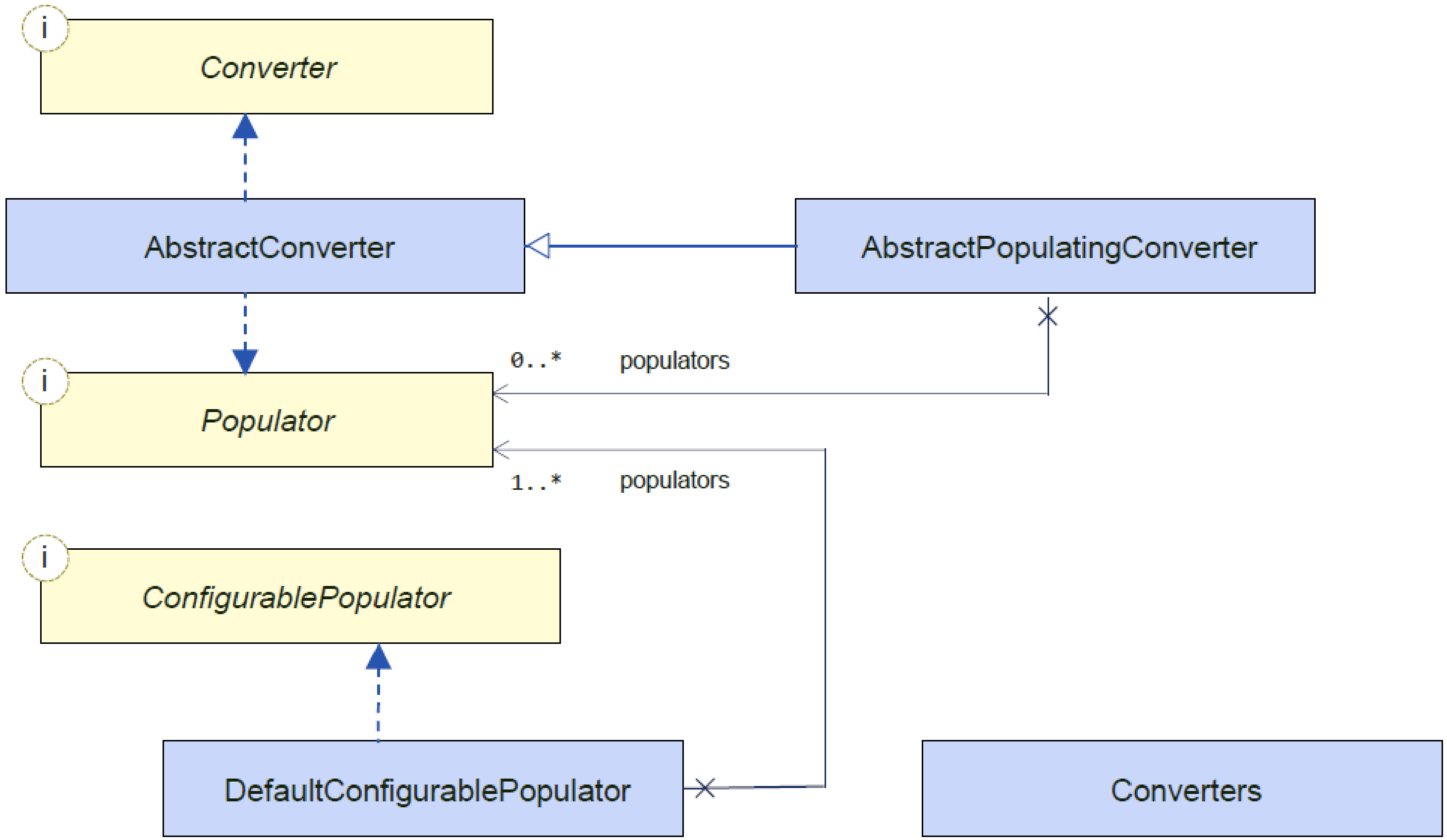
Configurable Populators

- Interface for a Populator that uses a collection of options to control what data is populated

Conceptual Interaction Diagram



Implementing classes



AbstractConverter:

- Base implementation which can be used as a converter and a populator
- Recommended to use Spring lookup-method for `createTarget`

Use case 1: Add a new Converter

Extend commercefacades with custom converters.

yacceleratorfacades-spring.xml

```
<alias name="defaultDeliveryModeConverter"
       alias="deliveryModeConverter"/>
<bean  id ="defaultDeliveryModeConverter"
       parent="abstractPopulatingConverter">
    <lookup-method name="createTarget" bean="deliveryModeData"/>
    <property name="populators">
        <list>
            <ref bean="deliveryModePopulator"/>
        </list>
    </property>
</bean>
```

Use case 2: Hook into existing types

How to hook properly into the type conversion to not rewrite the basic code or overwrite existing converters?

- `modifyPopulatorList` to modify existing populator lists
- defined in `commerceservices-spring.xml`
- Processed by BeanPostProcessor

`foofacades-spring.xml`

```
<bean parent="modifyPopulatorList">
    <property name="list" ref="productConverter"/>
    <property name="add"  ref="fooProductPopulator"/>
</bean>
```

Use case 3: Extended types and converters

How to hook properly into the type conversion with extended types?

```
<itemtype code="FooProduct" extends="Product" ...
<attribute qualifier="bar" type="java.lang.String" ...
```

Solution 1

- Write custom converter by extending the base type converter
- Decide at call level (Controller) which converter should be used

Solution 2

- Merge new attributes in ProductData (base type DTO)
- Add additional populator to modifyPopulatorList
- Populator must do an instance check on source type