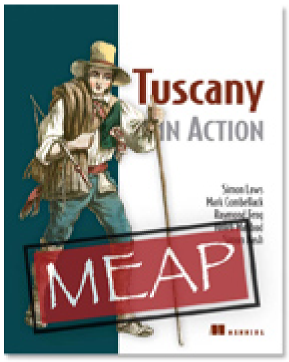
Defining a Java Component Implementation

Excerpted from



Tuscany in Action
EARLY ACCESS EDITION

Simon Laws, Mark Combellack, Raymond Feng, Haleh Mahbod and Simon Nash
MEAP Release: February 2009
Softbound print: Summer 2010; 400 pages
ISBN: 9781933988894

This article is taken from the book Tuscany in Action. *It looks at how to define a Java-based component implementation, and how Java annotations can be added to a Java class to define SCA services, references and properties.*

Each component in an SCA composite application is implemented using an implementation type. The SCA specifications define a number of implementation types and the Tuscany project has added a few more.

The SCA Java Component Implementation specification defines implementation.java (http://www.osoa.org/download/attachments/35/SCA_JavaComponentImplementation_V100.pdf). This implementation type allows application developers to use new or existing Java classes to implement SCA components. These components can then be wired with other components, either locally or remotely, to form a composite application. Let's start with the basics of how you define a Java based component implementation.

1 Defining a Java component implementation

Defining a Java component implementation is very simple. We'll use the Payment component from the TuscanySCATours application as an example. In its simplest form the Payment component is defined using the implementation.java element which references the Java class payment.PaymentImpl. The following snippet shows how the component is defined and made available in the payment composite.

```
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
           targetNamespace="http://scatours"
           name="payment">

  <component name="Payment">
      <implementation.java class="payment.PaymentImpl" />
  </component>

</composite>
```

Figure 1 shows how the Payment component Java implementation fits in with relation to SCA services, reference and properties.

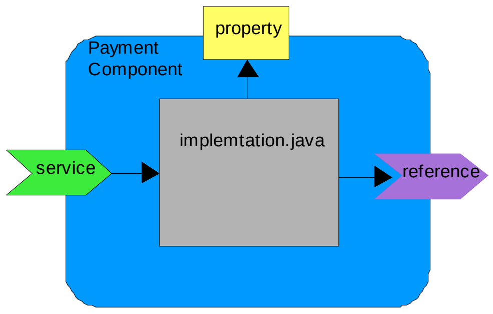


Figure 1: The Java implementation provides the business logic for a component, providing services and using references and properties

Of course, in the TuscanySCATours application, the Payment component collaborates with other components. Figure 2 shows a composite application that consists of the Payment component connected to other components each implemented using implementation.java.

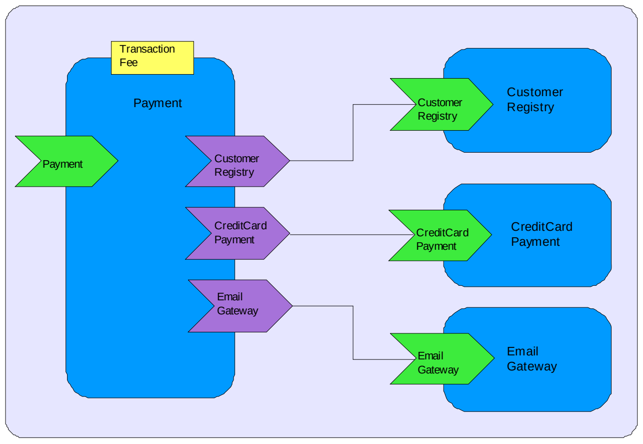


Figure 2: The Payment java component connected to the other components that it depends on

The CustomerRegistry component looks up customer payment information based on the customer's ID, the CreditCardPayment component handles the payment itself and the EmailGateway component notifies the customer of the payment status. The code snippet bellow shows the configuration of the Payment component as it appears in a composite file.

```
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
           targetNamespace="http://scatours"
           name="payment">

  <component name="Payment">
      <implementation.java class="payment.PaymentImpl" />
      <reference name="customerRegistry"
                 target="CustomerRegistryComponent"/>
      <reference name="creditCardPayment"
                 target="CreditCardPaymentComponent"/>
      <reference name="emailGateway"
                 target="EmailGatewayComponent"/>
      <property name="transactionFee”>0.02</property>
  </component>
```

```
      …
  </composite>
```

The target attribute of each reference element configures the named reference in the component implementation. The property element sets a value for the transactionFee property in the component implementation. Notice that we can change the reference (target or binding) and property (value) settings without changing the PaymentImpl class. The great advantage of this is that these decisions can be deferred until application assembly time.

Are you wondering how the services, references and properties are identified within the Java implementation? Let's move on and use the payment example to look at how Java annotations can be added to a Java class to define SCA services, references and properties for the Payment component. There are different styles you can use to do this.

2 Using SCA annotations in Java implementations

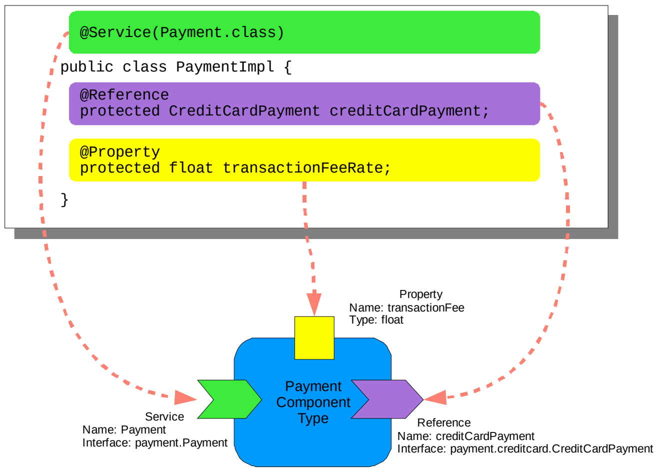
A component type describes the shape of a component in terms of the services it provides and the references and properties that it uses. SCA defines several Java annotations that allow the component implementation developer to describe the component type easily. The annotations are defined in the SCA Java Common Annotations and APIs specification (http://www.osoa.org/download/attachments/35/SCA_JavaAnnotationsAndAPIs_V100.pdf).

In our example the PaymentImpl class implements the Payment component. Listing 1 shows the source code for PaymentImpl, the annotations uses to identify services references and properties and how the annotated fields are used in the implementation.

Listing 1 Implementation class for the Payment component

```
@Service(Payment.class)
public class PaymentImpl implements Payment {
    @Reference
    protected CustomerRegistry customerRegistry;

    @Reference
    protected CreditCardPayment creditCardPayment;

    @Reference
    protected EmailGateway emailGateway;

    @Property
    protected float transactionFee = 0.01f;

    public String makePaymentMember(String customerId, float amount) {
      Customer customer = customerRegistry.getCustomer(customerId);
      String status = creditCardPayment.authorize(customer.getCreditCard(),
                                                  amount + transactionFee);
      emailGateway.sendEmail("order@tuscanyscatours.com",
                             customer.getEmail(),
                             "Status for your payment",
                             customer + " >>> Status = " + status);
      return status;
    }
}
```

Figure 3 gives a pictorial representation of how the PaymentImpl class uses SCA java annotations to define SCA service, references and properties. In this figure, @Service is used to describe Payment as an available service. The @Reference annotation is used to describe the dependency of the Payment component on the creditCardPament service. The @Property annotation is used to describe how the business logic can be altered depending on business need. The rest of the PaymentImpl class will contain pure business logic to implement the business process.



```
@Service(Payment.class)
public class PaymentImpl {
  @Reference
  protected CreditCardPayment creditCardPayment;

  @Property
  protected float transactionFeeRate;
}
```

Figure 3: Mapping the java implementation class to an SCA component type using SCA annotations

We now have the basic idea of how a java class can be used to implement an SCA component.