OpenXava is a framework for rapid development of business applications using Java. It is easy to learn and one can have an application up in no time. At the same time, OpenXava is extensible, customizable and the application code is structured in a very pure object oriented way, allowing you to develop arbitrarily complex applications.

The OpenXava approach for rapid development differs from those that use visual environments (like Visual Basic or Delphi) or scripting (like PHP). Instead, OpenXava uses a model-driven development approach, where the core of your application are Java classes that model your problem. This means you can stay productive while still maintaining a high level of encapsulation.

This chapter will show you the concepts behind OpenXava as well as an overview of its architecture.

1.1 The OpenXava concepts

Though OpenXava takes a very pragmatic approach to development, it is based on the refinement of two well known ideas: The very popular methodology of Model-Driven Development (MDD) and the concept of Business Component. Ideas from MDD are borrowed in a rather lightweight way. The Business Component, however, is at the very heart of OpenXava.

Let's look at these concepts in closer detail.

1.1.1 Lightweight Model-Driven Development

Basically, MDD states that only the model of an application needs to be developed, and that the rest is automatically generated. This is illustrated in figure 1.1.

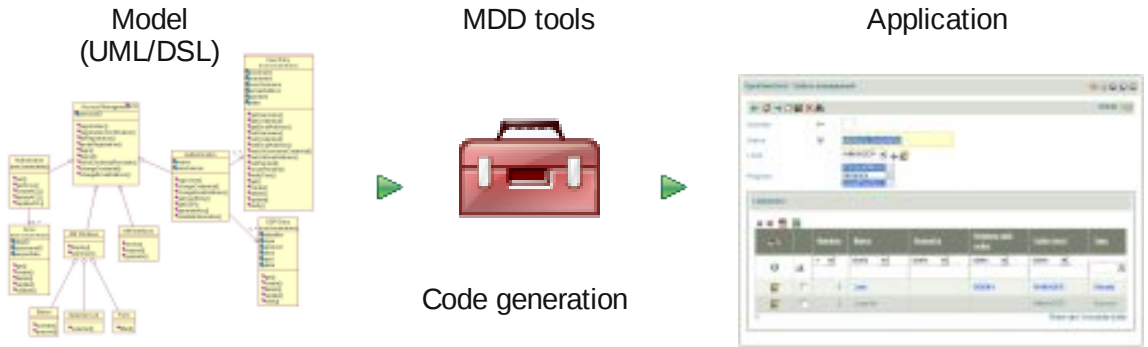

Figure 1.1 Model-Driven Development

In the context of MDD the model is the means of representing the data and the logic of the application. It can be either a graphical notation, such as UML, or a

textual notation such as a Domain-Specific Language (DSL).

Unfortunately, using MDD is very complex. It requires a big investment of time, expertise, and tooling[2]. Still the idea behind MDD is very good and hence OpenXava uses that idea in a simplified way. OpenXava uses plain annotated Java classes for defining the model, and instead of generating code, all functionalities are generated dynamically at runtime (table 1.1).

	Model definition	**Application generation**
Classic MDD	UML/DSL	Code generation
OpenXava	Simple Java classes	Dynamically at runtime

Tabla 1.1 MDD / OpenXava comparison

Figure 1.2 shows why we call OpenXava a *Lightweight Model-Driven Framework.*

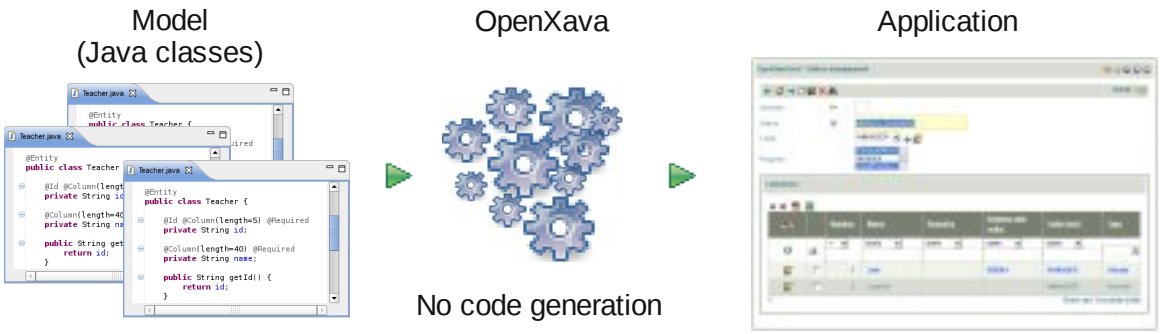

Figure 1.2 Lightweight Model-Driven Development in OpenXava

From just plain Java classes you obtain a full-fledged application. The next section about the Business Component concept will reveal some important details about the nature of these classes.

1.1.2 Business Component

A Business Component is a part of an application containing all software artifacts related to some business concept (e.g., an invoice), it is merely a way of organizing software. The orthogonal way of developing software is the paradigm of MVC (Model-View-Controller) where the code is compartmentalized by data (Model), user interface (View), and logic (Controller).

2 See the Better Software with Less Code white paper (http://www.lulu.com/content/6544428)

Figure 1.3 shows the organization of software artifacts in a MVC application. All code units associated with the creation of the user interface, like JSP pages, JSF, Swing, JavaFX, etc., are kept closely together in the view layer, and likewise, for the model and controller layers. This contrasts with a business component architecture where the software artifacts are organized around business concepts. This is illustrated in figure 1.4.

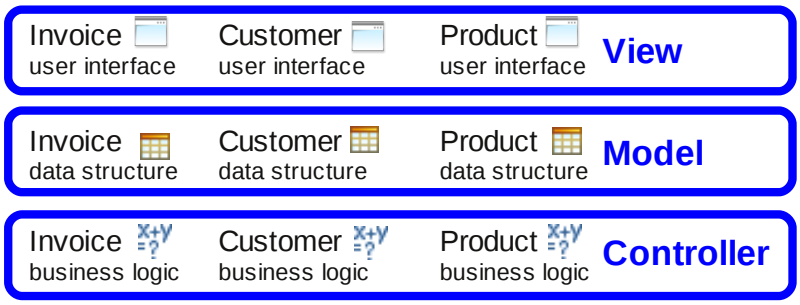

Figura 1.3 Enfoque Modelo Vista Controlador

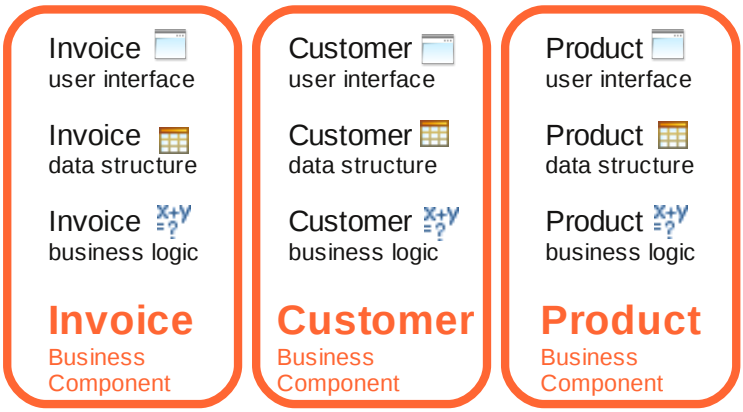

Figure 1.4 Business Component approach

Here, all software artifacts contributing to the Invoice concept, like user interface, database access, and business logic, are gathered in the same place.

Which paradigm to choose depends on your needs. If your data structures and the business logic are likely to change frequently, then the Business Component approach is very useful since all changes can be made in the same place instead of being scattered over multiple files.

In OpenXava the main part to develop is the Business Component, which is defined as a simple annotated Java class, exemplified in listing 1.1.

Listing 1.1 Invoice: A Java class for defining a business component

```
@Entity  // Database
@Table(name="GSTFCT")  // Database
@View(members=  // User interface
    "year, number, date, paid;" +
    "customer, seller;" +
    "details;" +
    "amounts [ amountsSum, vatPercentage, vat ]"
)
public class Invoice {

    @Id  // Database
    @Column(length=4)  // Database
    @Max(9999)  // Validation
    @Required  // Validation
    @DefaultValueCalculator(  // Declarative business logic
```

```
        CurrentYearCalculator.class
    )
    private int year;  // Data struture (1)

    @ManyToOne(fetch=FetchType.LAZY)  // Database
    @DescriptionsList    // User interface
    private Seller seller;  // Data struture

    public void applyDiscounts() {  // Programmatic business logic (2)
        ...
    }

    ...
}
```

As you can see, everything to do with the concept of an Invoice is defined in a single place: the `Invoice` class. This class contains code dealing with persistence, data structures, business logic, user interface, validation, etc.

This is accomplished using the Java metadata facility, so-called annotations. Table 1.2 shows the annotations used in this example.

Facet	Metadata	Implemented by
Database	@Entity, @Table, @Id, @Column, @ManyToOne	JPA
User interface	@View, @DescriptionsList	OpenXava
Validation	@Max, @Required	Hibernate Validator, OpenXava
Business logic	@DefaultValueCalculator	OpenXava

Table 1.2 Metadata (annotations) used in Invoice business component

Thanks to metadata you can do most of the work in a declarative way and the dirty work is done for you by JPA, Hibernate Validator and OpenXava.

Moreover, the code you write is plain Java, like properties (`year` and `seller`, 1) for defining the data structure, and methods (`applyDiscounts()`, 2) for programmatic business logic.

All you need to write about invoice is *Invoice.java*. It is a Business Component. The magic of OpenXava is that it transforms this Business Component into a ready to use application.

1.2 Application architecture

You have seen how Business Components are the basic cells to construct in an OpenXava application. Indeed, a complete OpenXava application can be created using only Business Components. Nevertheless, there are plenty of additional ingredients available.

1.2.1 Application developer viewpoint

As stated above, a fully functional application can be built using only Business Components. Usually, however, it is necessary to add more functionalities in order to fit the behavior of the application to your needs. A complete OpenXava application has the shape of figure 1.5.

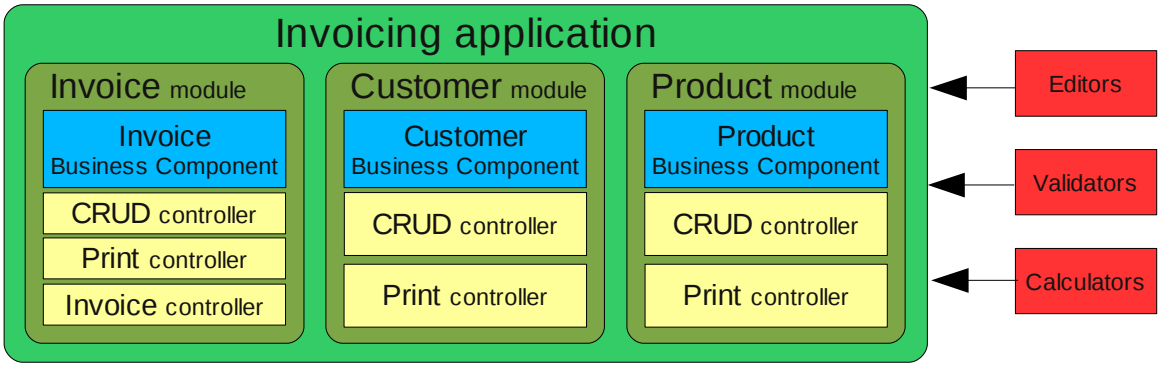

Figure 1.5 Shape of an OpenXava application

In figure 1.5, apart from Business Components, there are modules controllers, editors, validators and calculators. Let's see what these things are:

- **Business components**: Java classes that describe all aspects of the business concepts. These classes are only required pieces in an OpenXava application.
- **Modules**: A module is what the final user sees. It's the union of a Business Component and several controllers. You can omit the module definition, in which case a default module is used for each Business Component.
- **Controllers**: A controller is a collection of actions. From a user viewpoint, actions are buttons or links he can click; for the developer, they are the classes containing program logic to execute when those buttons are clicked. The controllers define the behavior of the application and can be reused in different modules of your application. OpenXava comes with a set of predefined controllers for many everyday tasks, and, of course, you can also define your own custom controllers.
- **Editors**: Editors are user interface components that specify how different members and attributes of the Business Component are displayed and edited. They provide a means for extending and customizing the user interface.
- **Validators**: Reusable validation logic that you can use in any Business Component.