CONTENTS INCLUDE:

Vaadin: A Familiar Way to Build Web Apps with Java

By Marko Grönroos

ABOUT VAADIN

Vaadin is a server-side Ajax web application development framework that allows you to build web applications just as you would with traditional desktop frameworks, such as AWT or Swing. An application is built from user interface components contained hierarchically in layout components.

In the server-driven model, the application code runs on a server, while the actual user interaction is handled by a client-side engine running in the browser. The client-server communications and any client-side technologies, such as HTML and JavaScript, are invisible to the developer. As the client-side engine runs as JavaScript in the browser, there is no need to install plug-ins.

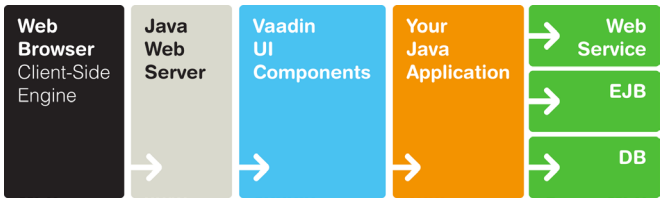

Figure 1: Vaadin Client-Server Architecture

If the built-in selection of components is not enough, you can develop new components with the Google Web Toolkit (GWT) in Java.

CREATING AN APPLICATION

An application that uses the Vaadin framework needs to inherit the **com.vaadin.Application** class and implement the `init()` method.

```
import com.vaadin.ui.*;

public class HelloWorld extends com.vaadin.Application {
    public void init() {
        Window main = new Window("Hello window");
        setMainWindow(main);
        main.addComponent(new Label("Hello World!"));
    }
}
```

To write an application class and the initialization method, you should:

- inherit the Application class
- create and set a main window
- populate the window with initial components
- define event listeners to implement the UI logic

Optionally, you can also:

- set a custom theme for the window
- bind components to data
- bind components to resources

The application can change the components and the layout dynamically according to the user input.

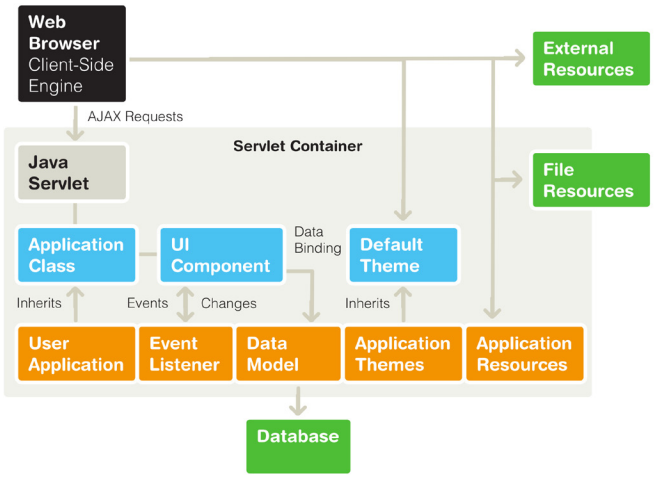

Figure 2: Architecture for Vaadin Applications

You can create a Vaadin application project easily with the Vaadin Plugin for Eclipse, with NetBeans, with Maven or with Spring Roo.

Event Listeners

In the event-driven model, user interaction with user interface components triggers server-side events, which you can handle with event listeners.

In the following example, we handle click events for a button with an anonymous class:

```
Button button = new Button("Click Me!");
button.addListener(new ClickListener() {
    public void buttonClick(ClickEvent event) {
        getWindow().showNotification("Thank You!");
    }
});
```


Vaadin: A Familiar Way to Build Web Apps with Java

Hot Tip You can embed Vaadin applications to any web page in `div` or `iframe` elements. See the *Book of Vaadin* for more info.

Below is a list of the most important event interfaces; their corresponding listener interface is named *-Listener*.

Event Interface	Description
Property.ValueChangeEvent	Field components except Button
Button.ClickEvent	Button click
Window.CloseEvent	A sub-window or an application-level window has been closed

Unless the immediate property (see below) is set, value change events are not communicated immediately to the server-side when the user changes the values. Instead, they are delayed until the first immediate interaction. Certain events, such as button clicks, are immediate by default.

Deployment

To deploy an application as a servlet, you must define a **WEB-INF/web.xml** deployment descriptor. The application class must be defined in the application parameter.

```
<web-app>
  <display-name>myproject</display-name>

  <servlet>
    <servlet-name>Myproject Application</servlet-name>
    <servlet-class>com.vaadin.terminal.gwt.server.ApplicationServlet
    </servlet-class>
    <init-param>
      <description>Vaadin application class to start</description>
      <param-name>application</param-name>
      <param-value>com.example.myproject.HelloWorld</param-value>
    </init-param>
  </servlet>

  <servlet-mapping>
    <servlet-name>Myproject Application</servlet-name>
    <url-pattern>/*</url-pattern>
  </servlet-mapping>
</web-app>
```

You can also deploy to a portlet.

COMPONENTS

Vaadin components include field, layout, and other components. The component classes and their inheritance hierarchy is illustrated in Figure 4.

Component Properties

Common component properties are defined in the **Component** interface and the **AbstractComponent** base class.

Property	Description
caption	A label that identifies the component for the user, usually shown above, left of, or inside a component, depending on the component and the containing layout.
description	A label that identifies the component for the user, usually shown above, left of, or inside a component, depending on the component and the containing layout.
enabled	If false, the user can not interact with the component. The component is shown as grayed. (Default: *true*)
icon	An icon for the component, usually shown left of the caption.
immediate	If *true*, value changes are communicated immediately to the server-side, usually when the selection changes or (a text field) loses input focus. The default is false for most components, but true for Button.
locale	The current country and/or language for the component. Meaning and use is application-specific for most components. (Default: application locale)
readOnly	If true, the user can not change the value. (Default: *false*)
visible	Whether the component is actually visible or not. (Default: *true*)

Field Properties

Field properties are defined in the **Field** interface and the **AbstractField** base class for fields.

Property	Description
required	Boolean value stating whether a value for the field is required. (Default: *false*)
requiredError	Error message to be displayed if the field is required but empty.

Sizing

The size of components is defined in the **Sizeable** interface.

Method	Description
`setWidth()` `setHeight()`	Set the component size in either fixed units (px, pt, pc, cm, mm, in, or em) or as a relative percentage (%) of the containing layout. The value "-1" means undefined size (see below).
`setSizeFull()`	Sets both dimensions to 100% relative size of the space given by the containing layout.
`setSizeUndefined()`	Sets both dimensions as undefined, causing the component to shrink to fit the content.

Notice that a layout with an undefined size must not contain a component with a relative (percentual) size.

Validation

All components implementing the **Validatable** interface, such as all fields, can be validated with `validate()` or `isValid()`. You need to implement a **Validator** and its `validate()` and `isValid()` methods and add the validator to the field with `addValidator()`.

Built-in validators are defined in the **com.vaadin.data.validator** package and include:

Validator	Description
DoubleValidator	A floating-point value
EmailValidator	An email address
IntegerValidator	An integer value
RegexpValidator	String that matches a regular expression
StringLengthValidator	Length of string is within a range

Resources

Icons, embedded images, hyperlinks, and downloadable files are referenced as *resources*.

```
Button button = new Button("Button with an icon");
button.setIcon(new ThemeResource("img/myimage.png"));
```

The external and theme resources are usually static resources. Application resources are served by the Vaadin application servlet, or by the user application itself.

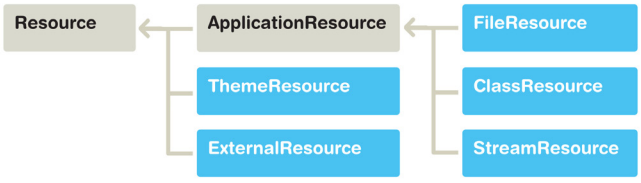

Figure 3: Resource classes and interfaces

Continued on Page 4...

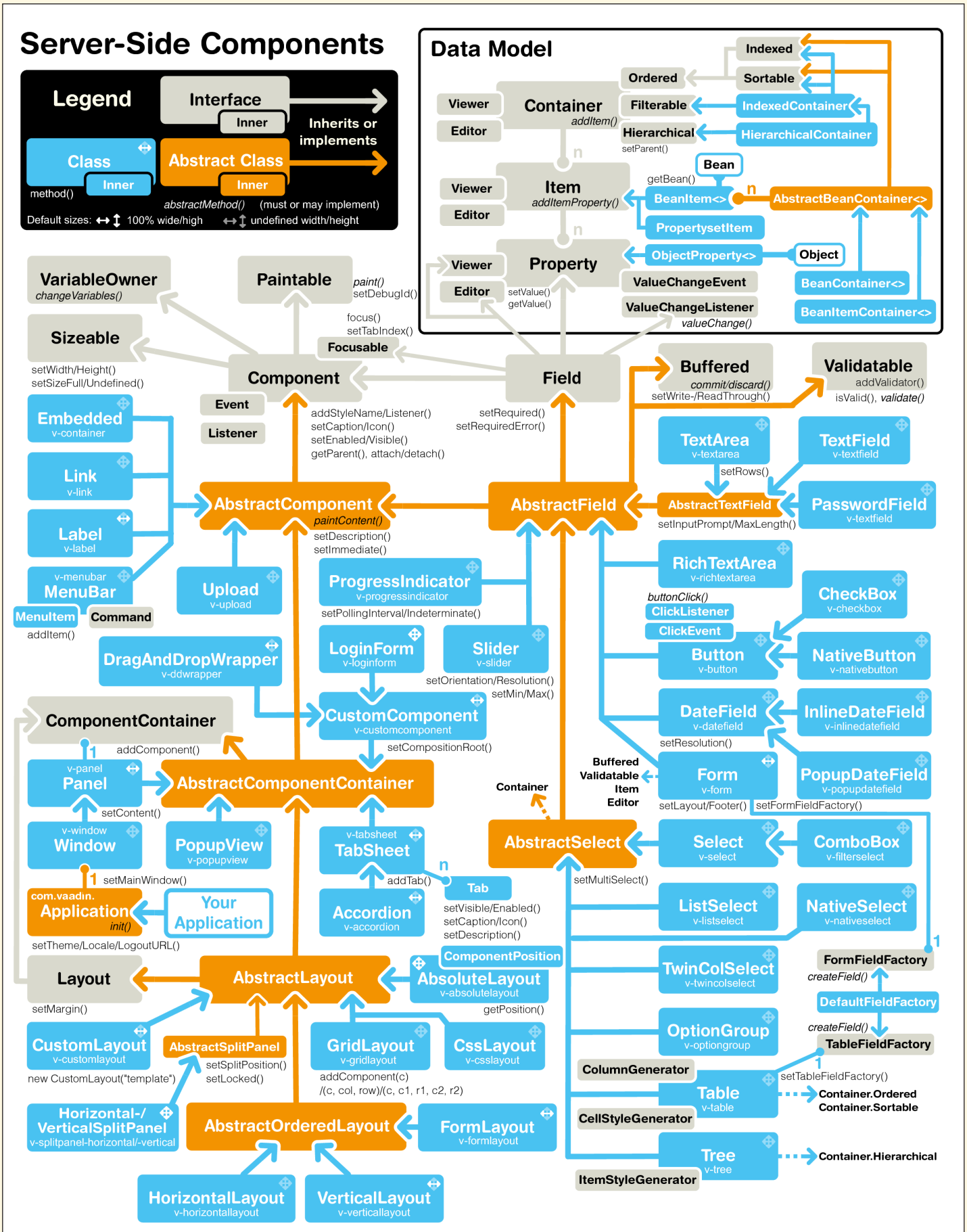

Figure 4: The Class Diagram presents all user interface component classes and the most important interfaces, relationships, and methods.

...Continued from Page 2

Class Name	Description
ExternalResource	Any URL
ThemeResource	A static resource served by the application server from the current theme. The path is relative to the theme folder.
FileResource	Loaded from the file system
ClassResource	Loaded from the class path
StreamResource	Generated dynamically by the application

LAYOUT COMPONENTS

The layout of an application is built hierarchically with layout components or more generally component containers.

You start by creating a root layout for the main window and set it with `setContent()`, unless you use the default, and then add the other layout components hierarchically with `addComponent()`.

Margins

The margin of layout components is controlled with the *margin* property, which you can set with `setMargin()`. Once enabled, the HTML element of the layout will contain an inner element with `<layoutclass>-margin` style, for example, `v-verticallayout-margin` for a **VerticalLayout**. You can use the *padding* property in CSS in a custom theme to set the width of the margin.

Spacing

Some layout components allow spacing between the elements. You first need to enable spacing with `setSpacing(true)`, which enables the `<layoutclass>-spacing-on` style for the layout, for example, `v-gridlayout-spacing-on` for **GridLayout**. You can then set the amount of spacing in CSS in a custom theme with the `padding-top` property for vertical and `padding-left` for horizontal spacing.

Alignment

When a layout cell is larger than a contained component, the component can be aligned within the cell with the `setComponentAlignment()` method.

Custom Layout

The **CustomLayout** component allows the use of a HTML template that contains location tags for components, such as `<div location="hello">`. The components are inserted in the location elements with the `addComponent()` method as shown below:

```
CustomLayout layout = new CustomLayout("mylayout");
layout.addComponent(new Button("Hello"), "hello");
```

The layout name in the constructor refers to a corresponding `.html` file in the `layouts` subfolder in the theme folder, in the above example `layouts/mylayout.html`. See Figure 5 for the location of the layout template.

Creating Composite Components

You can create composite components by extending layout components. The **CustomComponent** allows component composition while hiding the implementation details. You need to extend it and set the composition root, usually a layout component, with `setCompositionRoot()` in the constructor.

THEMES

Vaadin allows customization of appearance of the user interface with *themes*. Themes can include CSS style sheets, custom layout HTML templates, and any graphics.

Custom themes are placed under the `WebContent/VAADIN/themes/` folder of the web application. This location is fixed. The `VAADIN` folder specifies that these are static resources specific to Vaadin.

The name of the theme folder defines the name of the theme, to be used for the `setTheme()` method:

```
public void init() {
    setTheme("mytheme");
    …
```

The theme folder must contain the styles.css style sheet and custom layouts must be placed in the layouts sub-folder, but other contents may be named freely.

Custom themes need to inherit a base theme in the beginning of the styles.css file. The default theme for Vaadin 6 is reindeer.

```
@import url(../reindeer/styles.css);
```

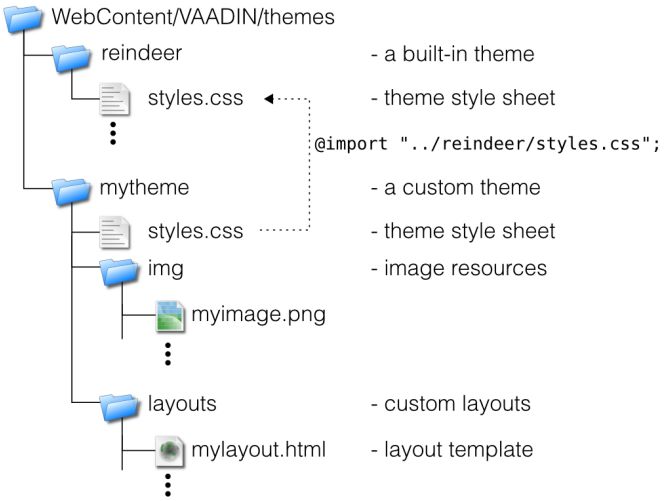

Figure 5: Theme contents

Hot Tip Try adding `?debug` or `?restartApplication` to the application URL.

DATA BINDING

Vaadin allows binding components directly to data. The data model, illustrated in Figure 4, is based on interfaces on three levels of containment: properties, items, and containers.

Properties

The **Property** interface provides access to a value of a specific class with the `setValue()` and `getValue()` methods.

All field components provide access to their value through the **Property** interface, and the ability to listen for value changes with a **Property.ValueChangeListener**. The field components hold their value in an internal data source by default but you can bind them to any data source with `setPropertyDataSource()`.

For selection components, the property value points to the item identifier of the current selection, or a collection of item identifiers in the *multiSelect* mode.

Items

An *item* is an ordered collection of properties. The **Item** interface also associates a name with each property. Common uses of items include **Form** data and **Table** rows.

The **BeanItem** is an adapter that allows accessing any Java bean (or POJO with the proper setters and getters) through the **Item** interface. This is useful for binding a **Form** to a bean.

Containers

A *container* is a collection of items. It allows accessing the items with an *item identifier* associated with each item.

Common uses of containers include selection components, as defined in the **AbstractSelect** class, especially the **Table** and **Tree** components. (The current selection is indicated by the property of the field, which points to the item identifier of the selected item.)

Vaadin includes the following container implementations:

Container Class	Description
IndexedContainer	Container with integer index keys
BeanItemContainer	Container for BeanItems
HierarchicalContainer	Tree-like container, used especially by the Tree component
FilesystemContainer	Direct access to the file system

Buffering

All field components implement the **Buffered** interface that allows buffering user input before it is written to the data source. Buffering is disabled by default.

Method	Description
`commit()`	Writes the buffered data to the data source
`discard()`	Discards the buffered data and re-reads the data from the data source
`set-/getWriteThrough()`	Set to `false` to enable write buffering
`set-/getReadThrough()`	Set to `false` to enable read buffering

USING ADD-ON COMPONENTS

In addition to the core features, you can find additional components, themes, container implementations, and various tools from the Vaadin Directory.

To install an add-on, download the Jar or Zip package from the Directory. If packaged as a Zip, extract the Jar library from it. Copy the Jar to the `WEB-INF/lib` path in your project.

Most add-on components contain a *widget set*. The widget sets from different add-ons need to be combined with the core widgets by compiling a *project widget set*. The widget set must be referenced in the `web.xml` descriptor.

- With Eclipse IDE, click the "Compile Vaadin Widgets" button in the toolbar. This requires the Vaadin Plugin for Eclipse.
- With NetBeans IDE and others, copy the build-widgetset.xml script template from the Vaadin installation package, edit it for your project, and run it with Ant.

More detailed instructions are given in the Directory (http://vaadin.com/directory) and *Book of Vaadin*.

CREATING NEW COMPONENTS

Vaadin components consist of a server-side component and a client-side GWT widget counterpart.

Creating a Client-Side Widget

To create a client-side component, you should:

- Implement the **Paintable** interface
- Maintain a reference to the **ApplicationConnection** object
- Implement `updateFromUIDL()` to deserialize state changes from server-side
- Serialize state changes to server-side with calls to `updateVariable()`

Creating a Server-Side Component

To create a server-side component, you should:

- Use `@ClientWidget` annotation for the server-side component class to bind the component to the client-side counterpart
- Implement `paintContent()` to serialize state changes to client-side with `addVariable()` and `addAttribute()` calls
- Implement `changeVariables()` to deserialize state changes from client-side

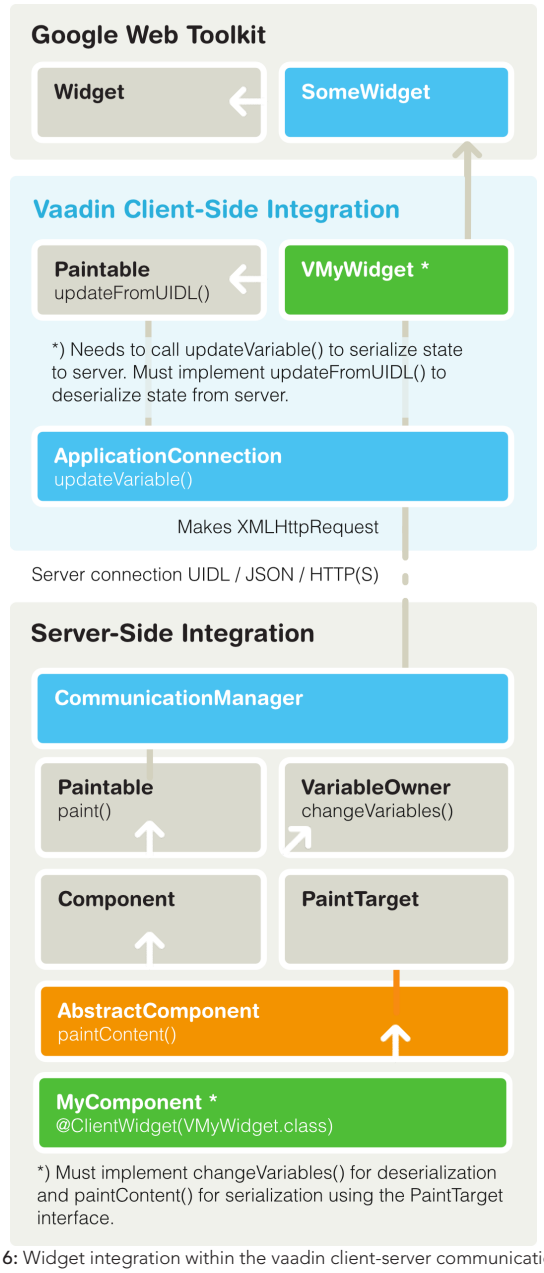

Figure 6: Widget integration within the vaadin client-server communication architecture

Defining a Widget Set

A widget set is a collection of widgets that, together with the communication framework, form the Client-Side Engine of Vaadin, when compiled with the GWT Compiler into JavaScript.

A widget set is defined in a `.gwt.xml GWT` Module Descriptor. You need to specify at least one inherited base widget set, typically the **DefaultWidgetSet** or a custom set.

```
<module>
    <inherits name="com.vaadin.terminal.gwt.DefaultWidgetSet" />
    <inherits name="com.example.widgetset.AnotherWidgetSet" />
</module>
```

The client-side source files must be located in the `client` sub-package under the package of the descriptor.

You can associate a stylesheet with a widget set with the `<stylesheet>` element in the `.gwt.xml` descriptor:

```
<stylesheet src="colorpicker/styles.css"/>
```

You can create new widgets easily with the Vaadin Plug-in for Eclipse.

Widget Project Structure

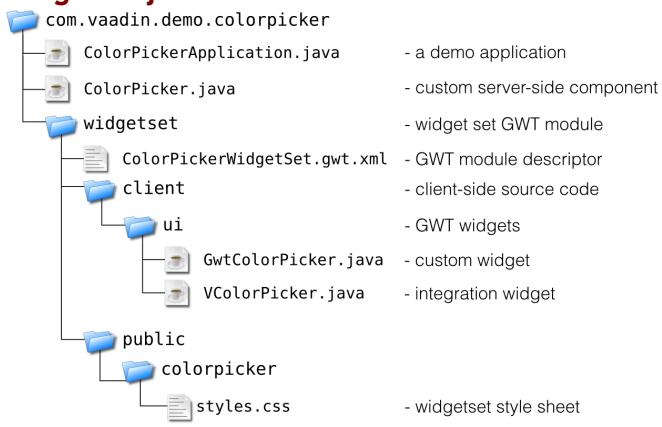

Figure 7: Widget set source structure.

The Figure 7 illustrates the source code structure of a widget project (for the Color Picker example).

For more information on Vaadin, visit the Vaadin Blog at http://vaadin.com/blog or the Forum at http://vaadin.com/forum

ABOUT THE AUTHOR

Marko Grönroos is a professional writer and software developer working at Vaadin Ltd, the company behind Vaadin. He has been involved in web application development since 1994 and has worked on several application development frameworks in C, C++, and Java. He has been active in many open-source software projects and holds an M.Sc. degree in Computer Science from the University of Turku. He lives in Turku, Finland.

Website: http://iki.fi/magi

Blog: http://markogronroos.blogspot.com/

RECOMMENDED BOOK

Book of Vaadin is a comprehensive documentation of Vaadin. It shows how to get started, gives a good overview of the features, and tutors you through advanced aspects of the framework.

READ NOW
http://vaadin.com/book

DZone

DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheat sheets, blogs, feature articles, source code and more. **"DZone is a developer's dream,"** says PC Magazine.

DZone, Inc.
140 Preston Executive Dr.
Suite 100
Cary, NC 27513

888.678.0399
919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com

Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-936502-40-0
ISBN-10: 1-936502-40-2
50795
9 781936 502400
$7.95

Version 1.0