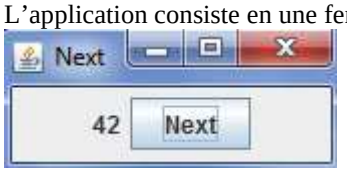
Conception orientée objet – Programmation orientée objet

Illustration des patrons de conception observateur et modèle/vue/contrôleur : Next

L'application consiste en une fenêtre composée d'une valeur et d'un bouton permettant de passer à la valeur suivante :

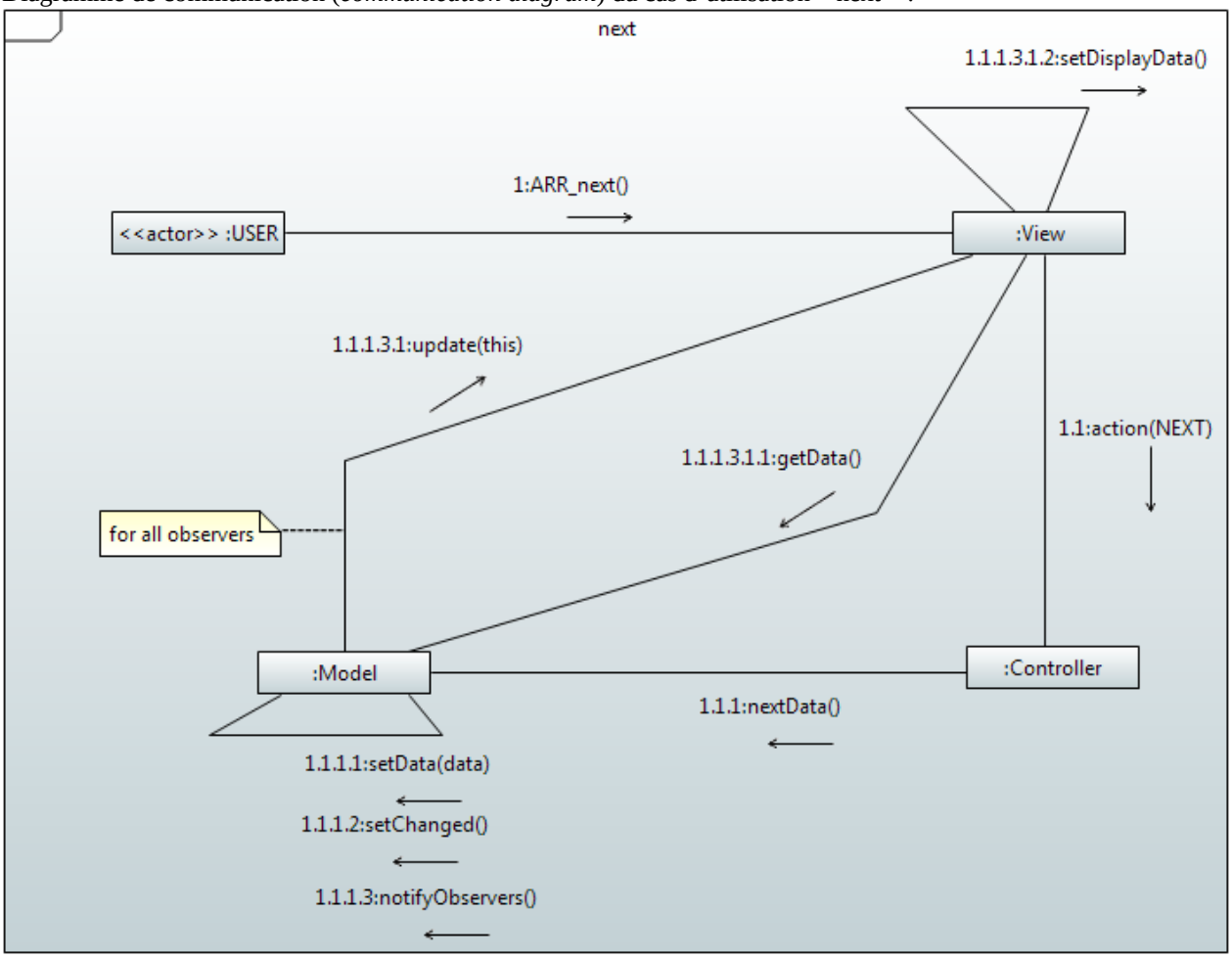


Il s'agit d'illustrer les patrons de conception (*design patterns*) observateur (*observer* (observateur) / *observable* (observé)) et modèle (*model*) / vue (*view*) / contrôleur (*controller*) ainsi que les liens entre différents diagrammes (cas d'utilisation, communication, classes ; ceux-ci ont été réalisés avec le *plugin* `Papyrus` d'`Eclipse`) d'*Unified Method Language*.

Les évolutions possibles sont : d'autres boutons (ex. : Previous), plusieurs vues (ex. : texte, graphique avec la valeur dans une barre de progression), plusieurs modèles, etc.

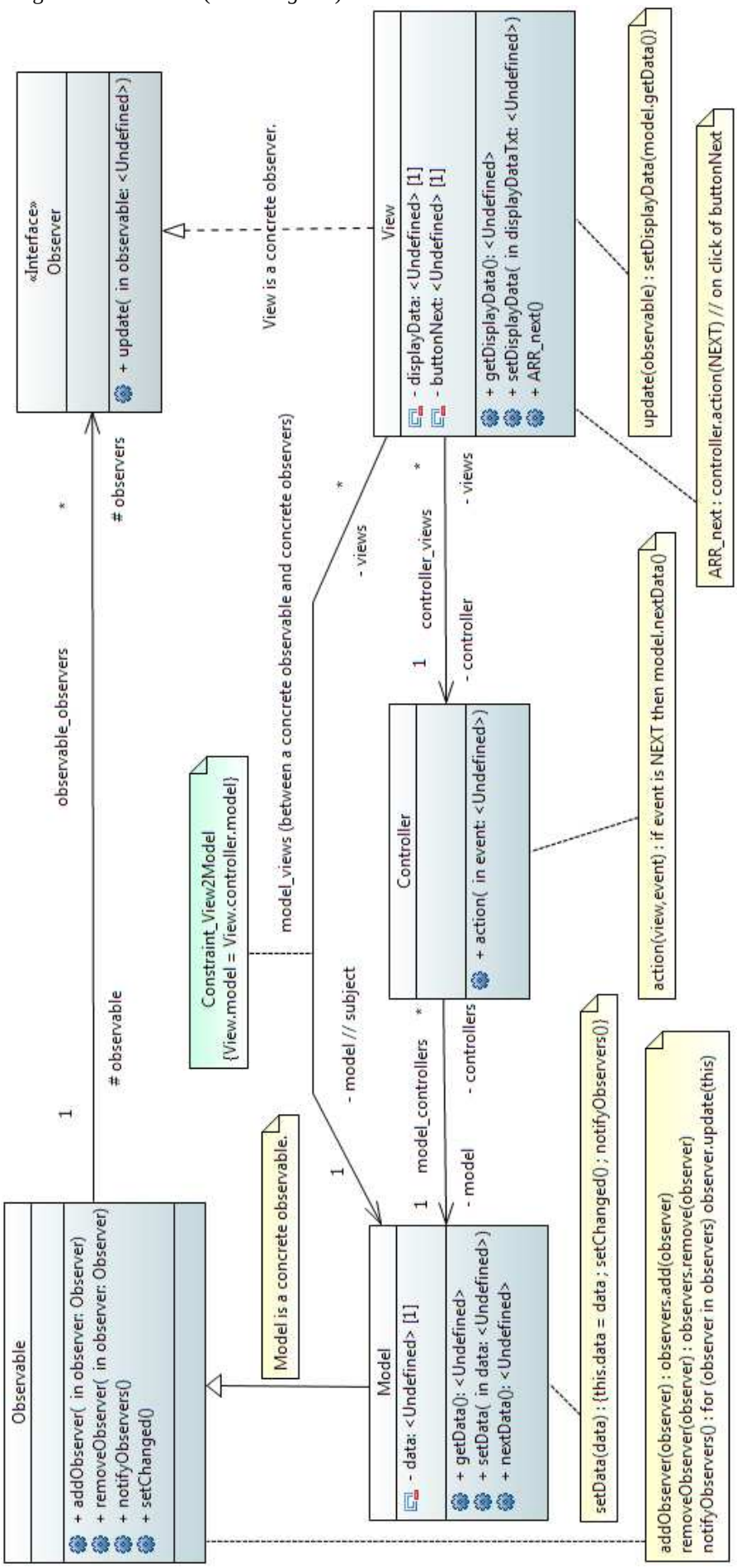
Diagramme des cas d'utilisation (*use case diagram*) :

Diagramme de communication (*communication diagram*) du cas d'utilisation « next » :



où `<<actor>> :USER` correspond à un acteur `:USER`, habituellement représenté par un *stickman*.

Diagramme de classes (*class diagram*) :



```java
/**
 * Application "Next".
 *
 * @author Olivier
 */

public class Next {

	/**
	 * Main of the application.
	 *
	 * @param args
	 *            Arguments.
	 */
	public static void main(String[] args) {
		// We must have view.model = view.controller.model.
		Model model = new Model();
		Controller controller = new Controller(model);
		View view = new View(controller, model);
		// Add the observer (the view) to be observable (by the model).
		model.addObserver(view);
	}

}
```

```java
import java.util.Observable;
import java.util.Random;

/**
 * Model.
 *
 * @author Olivier
 */
public class Model extends Observable {

	/**
	 * The default value of the data.
	 */
	private static final int DATA_DEFAULT = 42;

	/**
	 * The data.
	 */
	private int data;

	/**
	 * Create a model.
	 */
	public Model() {
		data = DATA_DEFAULT;
	}

	/**
	 * The data.
	 *
	 * @return The data.
	 */
	public int getData() {
		return data;
	}

	/**
```

```
	 * Set the data.
	 *
	 * @param data
	 *            The data to set.
	 */
	public void setData(int data) {
		this.data = data;
		setChanged();
		notifyObservers();
	}

	/**
	 * Next data.
	 */
	public void nextData() {
		Random rand = new Random();
		final int MIN_DATA = 01;
		final int MAX_DATA = 99;
		setData(rand.nextInt(MAX_DATA + 1 - MIN_DATA) + MIN_DATA);
	}

}
```

```
/**
 * Controller.
 *
 * @author Olivier
 */
public class Controller {

	/**
	 * The model.
	 */
	private Model model;

	/**
	 * Create a controller.
	 *
	 * @param model
	 *            The model.
	 */
	public Controller(Model model) {
		this.model = model;
	}

	/**
	 * Action to broadcast from the view to the model.
	 *
	 * @param event
	 *            Event.
	 */
	public void action(Event event) {
		switch (event) {
		case NEXT:
			model.nextData();
			break;
		default:
			;
		}
	}

}
```

```java
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Observable;
import java.util.Observer;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

/**
 * View.
 *
 * @author Olivier
 */
public class View extends JFrame implements Observer {

	/**
	 * Version number of the serializable class; unused!
	 */
	private static final long serialVersionUID = 1L;

	/**
	 * The controller.
	 */
	private Controller controller;

	/**
	 * The model.
	 */
	private Model model;

	/**
	 * The data to display.
	 */
	private JLabel displayData;

	/**
	 * The button to click if the display have to be changed.
	 */
	private JButton buttonNext = new JButton("Next");

	/**
	 * Create a view.
	 *
	 * @param controller
	 *            The controller.
	 * @param model
	 *            The model.
	 */
	public View(Controller controller, Model model) {
		super("Next");
		this.controller = controller;
		this.model = model;
		displayData = new JLabel(Integer.toString(this.model.getData()));
		setBounds(0, 0, 175, 75);
		JPanel panel = new JPanel();
		add(panel);
		panel.add(displayData);
		panel.add(buttonNext);
		buttonNext.addActionListener(new ActionListener() {
```

```
                public void actionPerformed(ActionEvent e) {
                        ARR_next(); // on click
                }
        });
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setVisible(true);
    }

    /**
     * The data to display.
     *
     * @return The data to display.
     */
    public JLabel getDisplayData() {
        return displayData;
    }

    /**
     * Set the data to display.
     *
     * @param displayDataTxt
     *            The text of the data to display to set.
     */
    public void setDisplayData(String displayDataTxt) {
        displayData.setText(displayDataTxt);
    }

    /**
     * The user wants the next data to display.
     */
    public void ARR_next() {
        controller.action(Event.NEXT);
    }

    @Override
    public void update(Observable arg0, Object arg1) {
        setDisplayData(Integer.toString(model.getData()));
    }

}
```

```
/**
 * Events.
 *
 * @author Olivier
 */
public enum Event {

    /**
     * Next.
     */
    NEXT;
}
```