Création des menus

Ref : TV110415_TU_SM

Rédacteur : Serge Morvan

0.1 Objet

Le but de cette section est d'expliquer la façon de rajouter un menu et ses sous-menus au Dock. Un certain nombre de services doivent être activés, ou désactivés par l'utilisateur, pour cela, le menu du Dock est privilégié. Le service à activer devra faire partie du groupe des `Driver` :

- `Driver` : pour l'ouverture de fichiers
- `DDriver` : pour le parcours de répertoires
- `InstrumentDriver` : pour le choix d'un instrument
- `DatabaseDriver` : pour la connexion à une base de données
- `WebDriver` : pour la connexion à un site web via un URL
- ...autres à venir

Ce service doit être enregistré dans le module `navisu-launcher` :

```
InstrumentDriverManagerServices instrumentDriverManagerServices =
        componentManager.getComponentService(InstrumentDriverManagerServices.class);
instrumentDriverManagerServices.init();
instrumentDriverManagerServices.registerNewDriver(sonarServices.getDriver());
instrumentDriverManagerServices.registerNewDriver(radarServices.getDriver());
```

Ensuite deux modules sont concernés, le module `navisu-app` pour spécifier le nouvel item dans le Dock (icônes et code) et le module `navisu-widgets` (icônes) car les `RadialMenu` sont des widgets. Dans les deux cas les images sont placées dans les ressources, correspondant à l'arborescence des classes `DockManagerImpl` et `RadialMenu` respectivement. Dans la suite nous prendrons l'exemple de l'instanciation, du démarrage ou de l'arrêt d'un `Instrument`. Bien entendu cet `Instrument` doit être implémenté au préalable.

```
public class SonarImpl
                implements Sonar, SonarServices, InstrumentDriver, ComponentState {
```

0.2 Le graphisme

0.2.1 Module `navisu-app`

Dessiner l'icône de l'item : à placer dans :
`bzh/terrevirtuelle/navisu/app/guiagent/dock/impl/dock_icons` des ressources

FIGURE 1 – *L'item Instruments*

0.2.2 Module `navisu-widgets`

Dessiner les icônes de l'item central du `RadialMenu` et de ses sous-items : à placer dans :
`bzh.terrevirtuelle.navisu.widgets.radialmenu.menu` des ressources.

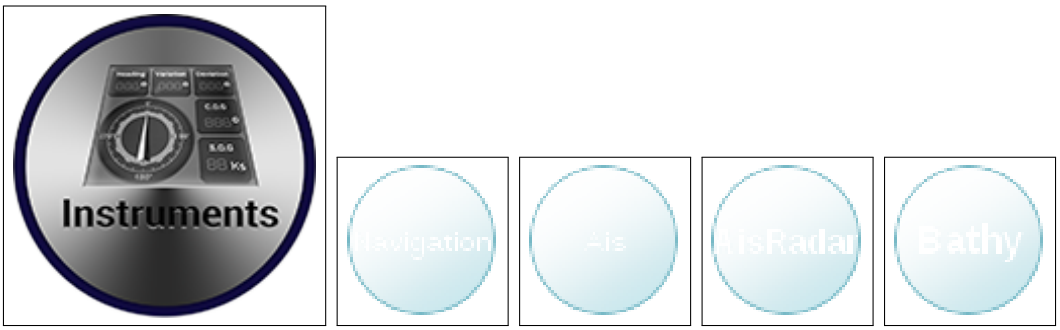

FIGURE 2 – *L'icône centrale de Instruments et les sous-items*

0.3 Le code

0.3.1 Module `navisu-app`

Dans la classe `DockManagerImpl` du module `navisu-app`, ajouter un item au Dock :

```
public final DockItem[] ICONS = new DockItem[]{
DockItemFactory.newImageItem("instruments",
     ICON_PATH + "dock_icons/instruments.png",
     (e) -> {
         instrumentsRadialMenu.setVisible(!instrumentsRadialMenu.isVisible());
     }),
     ...
```

Ici "instruments" correspond à l'infobulle associée,
ICON_PATH+"dock_icons/instruments.png" est le chemin de l'image dans le répertoire resources. Le dernier argument du constructeur est le callback associé, généralement celui là.
Appeler la méthode de création du RadialMenu associé :

```
@Override
public void makeDock() {
    createDockWidget(scene);
    createBooksRadialWidget();
    createChartsRadialWidget();

    createInstrumentsRadialWidget();

    createMeteoRadialWidget();
    createTidesRadialWidget();
    createToolsRadialWidget();
    createNavigationRadialWidget();
}
```

Coder cette méthode : le menu radial est créé à l'aide d'un RadialMenuBuilder

```
//--------------INSTRUMENTS------------------
private void createInstrumentsRadialWidget() {
    instrumentsRadialMenu = RadialMenuBuilder.create()
    .centralImage("instrumentsradialmenu.png")
    .createNode(0, "navigation.png", 0, "ais.png", 0, "aisradar.png",
                                                (e) -> open("AisRadar"))
    .createNode(0, "navigation.png", 1, "bathy.png", 0, "sonarOn.png",
                                                (e) -> open("Sonar"))
    .build();

    instrumentsRadialMenu.setLayoutX((width / 2) - 40);
    instrumentsRadialMenu.setLayoutY(height / 2);
    root.getChildren().add(instrumentsRadialMenu);

    radialMenus.add(instrumentsRadialMenu);
}
```

- La méthode createNode va préciser pour chaque item, son placement, ses images associées et son callback.
  - Un choix d'ergonomie a été fait, les menus radiaux ont deux couches de sous-menus puis des feuilles, les feuilles correspondent aux actions.
  - Dans la première couche les segments sont numérotés : 0, 1, 2, ... Dans l'exemple un seul segment, son icône est navigation.png"
  - Dans la deuxième couche, idem, les segments sont numérotés : 0, 1, 2, ...Dans

l'exemple deux segments, les icônes `ais.png` et `bathy.png`
— Chaque segment reçoit les feuilles correspondant aux items, ici un item par segment. les images `aisradar.png` et `sonarOn.png` respectivement.
— Enfin, le dernier argument correspond au callback associé à cet item.

FIGURE 3 – *Le menu Instruments*

0.3.2 Choix des `callbacks`

La plupart du temps les callbacks associés aux items n'ont pas besoin d'être codés, il sont fournis par NAVISU. Ci après la liste non exhaustive des calbacks et des cas d'utilisation.
— Cas d'un instrument : l'argument correspond au nom de l'instrument à activer.

```
private void open(String keyName) {
    instrumentDriverManagerServices.open(keyName);
    clear();
}
```

— Cas d'ouverture d'un fichier : le premier argument est le `KEY_NAME` du `Driver` sachant interpréter ces fichiers : Sedimentology, Currents, ...Le deuxième argument le ou les extensions des fichiers : `.shp, .SHP` ou `.000` par exemple.

```
private void open(String description, String... des) {
    String[] tab = new String[des.length];
    int i = 0;
    for (String s : des) {
        tab[i] = "*" + s;
        i++;
    }
    driverManagerServices.open(description, tab);
    clear();
}
```

— Connexion à une base de données :

```
private void openDB(String dbName, String hostName, String protocol, String port,
String driverName, String userName, String passwd) {
    databaseDriverManagerServices.connect(dbName, hostName, protocol,
                                          port, driverName, userName, passwd);
    clear();
}
```

NaVisu gère autant de bases de données qu'il est nécessaire, elles doivent être installées au préalable, sauf pour une base embarquée. C'est l'API JDBC qui est exhibée comme un ensemble de services.

0.3.3 Module `navisu-widgets`

Pas de code à écrire, il faut simplement placer les images correspondant au menu et à ses items, dans le fichier `resources`.

0.4 Principe des `DriverManager`

Comme il a été dit dans la préface, les services accessibles à partir du menu doivent s'enregistrer, voici l'explication. Le premier principe respecté est l'inversion de dépendance : la logique de l'application, contenue dans le module `navisu-app` ne doit pas dépendre des modules quelle active :

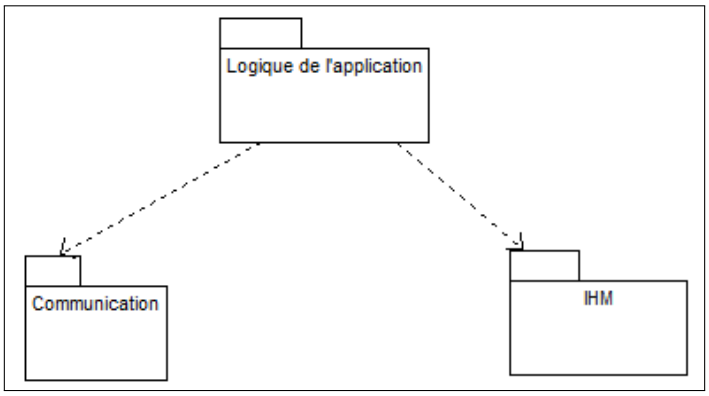

FIGURE 4 – *Une modification de l'IHM va impacter l'application*

La logique de l'application doit s'appuyer sur des services :

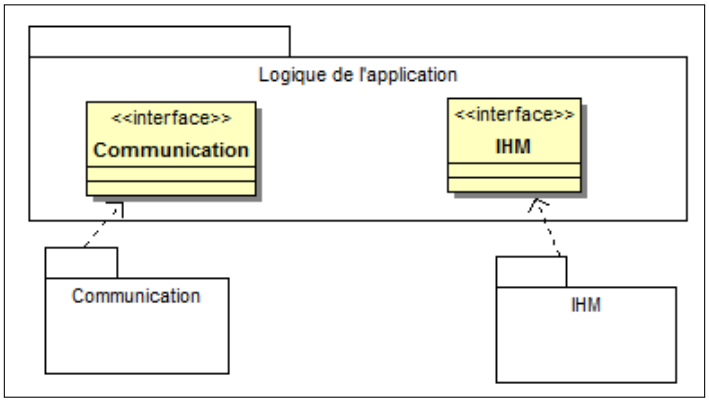

FIGURE 5 – *Si les contrats de services sont respectés, un changement du code de l'IHM ne concerne pas l'application*

Le deuxième principe est celui de pas de dépendances cycliques :

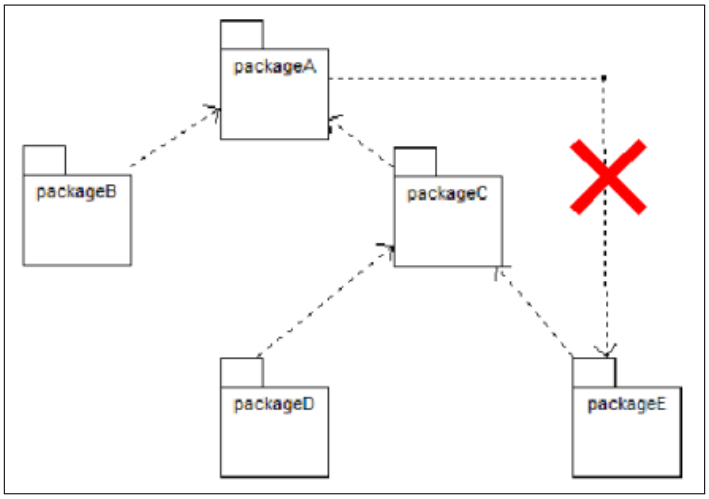

FIGURE 6 – *Gradle détectera un tel problème*

Dans chaque fichier `build.gradle` on trouve les dépendances du module, par exemple dans le sous-projet `navisu-intruments`

```
dependencies {
    compile project(':navisu-core')
    compile project(':navisu-client')
    compile project(':navisu-domain')
    compile project(':navisu-app')
    compile project(':navisu-bathymetry')
    compile fileTree(dir: 'lib', include: '*.jar')
}
```

Si le module `navisu-app` devait instancier un `Intrument` directement, il devrait connaitre, donc dépendre du module `navisu-intruments` ce qui constitue une rupture du principe numéro deux. Au lieu de cela les différents `Driver` s'inscrivent auprès de leur `DriverManager` spécifique, ensuite lors d'une requête lancée par les callbacks du dock, le `Driver` répondant aux critères de sélection est activé.

```
// class InstrumentDriverManagerImpl

protected List<InstrumentDriver> availableDriverList = new ArrayList<>();

@Override
public void registerNewDriver(InstrumentDriver driver) {
    Checker.notNull(driver, "Driver must not be null.");
    this.availableDriverList.add(driver);
}

@Override
public void open(String category) {
    InstrumentDriver driver = findDriver(category);
    if (driver != null) {
        driver.on();
    }else{
        System.out.println("Unrecognized instrument");
    }
}

protected InstrumentDriver findDriver(String category) {
    InstrumentDriver compatibleDriver = null;
    for (InstrumentDriver driver : this.availableDriverList) {
        if (driver.canOpen(category)) {
            compatibleDriver = driver;
            break;
        }
    }
    return compatibleDriver;
}
```

Exemple pour la classe `Sonar` les méthodes répondant au `DriverManager`

```
// public class SonarImpl
              implements Sonar, SonarServices, InstrumentDriver, ComponentState {

 private final String NAME = "Sonar";

@Override
public boolean canOpen(String category) {
    return category.equals(NAME);
}
@Override
    public void on() {
        ...
}
```