Tuto5: Developing Data Visualizations with React and D3.js

Nicolas Médoc Luxembourg Inst. of Science and Technology

Outline

1. Separation of responsibilities: React component + D3 class
2. React useEffect() hook
3. React useRef() hook
4. Creation of a scatterplot visualization

Combining D3.js and React: Separation of responsibilities

- **D3.js visualizations** are implemented in **self-contained class**, without any dependencies to React library
- For each visualization, **one React component** implements the **container of the visualization** and **makes the connection** between the D3js class and the React application.
- The React component:
  - **receives the data from the parent** through props propagation;
  - **instantiates the D3.js class** and call render/update methods to build/update the visualization,
  - **provides the event handler methods to D3.js** class to update the store.

Combining D3.js and React: ready-to-use templates

- `https://github.com/nicolasmedoc/Tuto5-MultiDim.git`
- Vis-d3.js
  - is a **javascript class** which will renders the visualization. It is self-contained and independent of React
  - declares a method **create()** to initialize the SVG element
  - declares a method **clear()** to remove the SVG from the DOM
  - declares **one or several update methods** (e.g. renderVis(), updateFunction1(), unpdateFunction2()) to change the view or a part of the visualization when the data changes, with the **global update pattern**.

Combining D3.js and React: first example

- VisContainer.js (React component)
  - is the container of the visualization.
  - controls the component lifecycle with **useEffect() hook** when the component **did mount**, **did update** (when data changes) or **did unmount** (when removed from the page)
  - **instantiates the d3 class** and store it in a **Ref** (with useRef() hook) to keep the d3 instance when the component re-render.
  - **renders the ** element containing the SVG and **stores it in a Ref** (with useRef() hook) to avoid re-render it.

React useEffect() hook to control the component life cycle

3 functions are declared in 2 different profiles of the useEffect() hook to declare additional behavior in the React component life cycle:

```
import { useEffect } from 'react';
...
useEffect(()=>{
    // the behavior after the component creation (did mount)
    return ()=>{
        // the behavior after the component deletion (did unmount)
        // is declared in the return function
    }
}, []) // empty array
useEffect(()=>{
    // behavior after update of dependency1 or dependency2 only
},[dependency1,dependency2]) // array of dependency variables
useEffect(()=>{
    // behavior after update of dependency3 only
},[dependency3]) // array of dependency variables
```

dependency1, **dependency2** and **dependency3** are variables derived from the state or propagated in props by the parent

React useRef() hook to persist a value in the component

Used to persist the instance object of the D3 class in a React component:

```
import { useEffect, useRef } from 'react';
...
const divContainerRef=useRef(null);
const matrixD3Ref = useRef(null)
useEffect(()=>{
    const matrixD3 = new MatrixD3(divContainerRef.current);
    matrixD3Ref.current = matrixD3;
},[])
```

React useRef() hook to persist a value in the component

Used to persist the previous value in the sub part of a dataSlice propagated in props:

```
import { useEffect, useRef } from 'react';
...
const dataSliceAttributeRef = useRef(null)
useEffect(()=>{
    // behavior after update of dataSlice
    if(dataSliceAttributeRef.current!==dataSlice.attribute){
        // attribute has been updated => do something
        // e.g. call specific update method in D3 class
        dataSliceAttributeRef.current = dataSlice.attribute
    }
},[dataSlice]) // array of dependency variables
```

Building a scatterplot: getting the data set

See in App.js the useEffect() hook function, how to retrieve data from data/Housing.csv, and store it in the state when the component is mounted

```
useEffect(()=>{
    console.log("App did mount");
    fetchCSV("data/Housing.csv",(response)=>{
        console.log("initial setData() ...")
        setData(response.data);
    })
    return ()=>{
        console.log("App did unmount");
    }
},[])
```

Building a scatterplot: render the visualization component

In ScatterplotContainer.js call the scatterplotD3.renderScatterplot() method in the useEffect hook handling data updates:

```
// get the current instance of scatterplotD3 from the Ref object...
const scatterplotD3 = scatterplotD3Ref.current
scatterplotD3.renderScatterplot(scatterplotData,xAttribute, yAttribute,
controllerMethods);
// controllerMethods being already declared
// with empty handleOnClick, handleOnMouseEnter and handleOnMouseLeave
```

Building a scatterplot: render the visualization component

Then, add the component in the rendered JSX of App.js.

```
const scatterplotControllerMethods= {
    // we will add here the update methods for interaction
};

return (
    <div className="App">
        <div id={"MultiviewContainer"} className={"row"}>
            <ScatterplotContainer scatterplotData={data}
                xAttribute="area"
                yAttribute="price"
                scatterplotControllerMethods={scatterplotControllerMethods}
            />
        </div>
    </div>
);
```

Building a scatterplot: creation of scales and X/Y axis

in components/scattterplot/Scatterplot-d3.js, in the method updateAxis():

Exercice1: using d3.min(mylist) and d3.max(mylist), set the domain values of this.xScale.domain(...) and this.yScale.domain(...) to put **"area" in X Axis and "price" in Y axis**.

In "updateAxis" function, use these scales to build X axis and Y axis (.xAxisG and .yAxisG are initialized in create() function):

```
this.matSvg.select(".xAxisG")
    .transition().duration(500)
    .call(d3.axisBottom(this.xScale))
;
this.matSvg.select(".yAxisG")
    .transition().duration(500)
    .call(d3.axisLeft(this.yScale))
```

Building a scatterplot: update circle positions with scales

in components/scattterplot/Scatterplot-d3.js, in the method updateMarkers():
Exercice2: using X/Y scales, apply a translation to .markerG to update the circle positions. updateMArkers(selection) is called from renderScatterplot(). The "selection" parameter comes from the update pattern using join(). It corresponds to a selection of ".markerG".

```
selection
    .transition().duration(500)
    .attr("transform", (itemData)=>{
        // use scales to return shape position from data values
})
```

Building a scatterplot: reuse the scatterplot component

Exercice 3: display two scatterplot side by side to show different pair of dimensions:

- scatterplot 1: (x="area" y="price")
- scatterplot 2: (x="bedrooms" y="price")

Coordinated multiple views: principles

- **Exercise 4:** synchronize the two scatterplots on click interaction. The purpose is to highlight the clicked objects simultaneously in the two scatterplots (e.g. make the red border visible).
- **In ScatterplotContainer.js useEffect()**: the object controllerMethods is propagated to the ScatterplotD3 class and contains all the methods defining the behaviour of interactions (handleOnClick, handleOnMouseEnter, handleOnMouseLeave)
- These methods have to **update the data states in the parent components** to synchonize the two scatterplots by calling methods propagated by the parent in the component props (scatterplotControllerMethods).

Coordinated multiple views: prepare the update methods in App.js

In App.js add the state for the synchronization and add the update method in the scatterplotControllerMethods propagated to the child component.

```
const [selectedItems, setSelectedItems] = useState([])

const scatterplotControllerMethods= {
    updateSelectedItems: (items) =>{
        setSelectedItems(items);
    }
};
```

Coordinated multiple views: use the update method in handleOnClick

In ScatterplotContainer.js call the update method in handleOnClick, which is propagated to the D3 class.

```
const handleOnClick = function(cellData){
    console.log("handleOnClick ...")
    scatterplotControllerMethods.updateSelectedItems([cellData])
}
```

Coordinated multiple views: highlight an object in the views

Propagate the synchronization data in props:

- add selectedItems as props in ScatterplotContainer
- propagate the selectedItems state from App.js to ScatterplotContainer (in the two rendered instances)

Coordinated multiple views: highlight an object in the views

In ScatterplotContainer, create a specific useEffect with selectedItems as dependency to call an update method in D3 class that will highlight the selected marker:

```
useEffect(()=>{
        console.log("ScatterplotContainer useEffect with dependency [selectedItems], " +
            "called each time selectedItems changes...");
    // get the current instance of scatterplotD3 from the Ref object...
    // call highlightSelectedItems of ScatterplotD3...;
},[selectedItems])
```

Coordinated multiple views: highlight an object in the views

In ScatterploContainer.js highlightSelectedItems method, use the update pattern to declare the right behaviour:

```
highlightSelectedItems(selectedItems){
    // use pattern update to change the border and opacity of objects:
    //      - call this.changeBorderAndOpacity(selection,true)
    //        for markers that match selectedItems
    //      - this.changeBorderAndOpacity(selection,false) for
    //        markers that do not match selectedItems
}
```

Coordinated multiple views: highlight an object in the views

By looking at the logs when clicking an item, the scatterplot renders again while it is not necessary. This is due to the various depencies declared in the useEffect, including the object "scatterplotControllerMethods" having a new instance at each React life cycle. **To render only when the scatterplotData changes, we use Ref object** to keep the previous state of scatterplotData and compare with the new state:

```
const scatterplotDataRef = useRef(scatterplotData);
// did update, called each time dependencies change, dispatch remain stable over component cycles
useEffect(()=>{
    console.log("ScatterplotContainer useEffect when  dependency [scatterplotData ...
    ...
    if(scatterplotDataRef.current != scatterplotData) {
        console.log("ScatterplotContainer useEffect when scatterplotData changes...");
        // get the current instance of scatterplotD3 from the Ref object...
        const scatterplotD3 = scatterplotD3Ref.current
        // call renderScatterplot of ScatterplotD3...;
        scatterplotD3.renderScatterplot(scatterplotData, xAttribute, yAttribute, controllerMethods);
        scatterplotDataRef.current = scatterplotData;
    }
    ...
```