Gestión dinámica de componentes QML – III

Adentrémonos en la versatilidad de QML para la animación de los interfaces de usuario.

Estados (States)

Un estado "**State**" es el **estado** en el que se encuentra el interfaz del usuario GUI en un momento del tiempo. Ejemplos: Al inicializar un elemento, antes de pulsar, después de pulsar, etc…

Los **estados** se aplican a la solución de muchos problemas diferentes de distinta complejidad.

```
Rectangle {
    id: rect
    width: 200
    height: 200
}
```

```
states: State {
    name: "shorter"
    PropertyChanges {
        target: rect;
        height: 100
    }
}
```

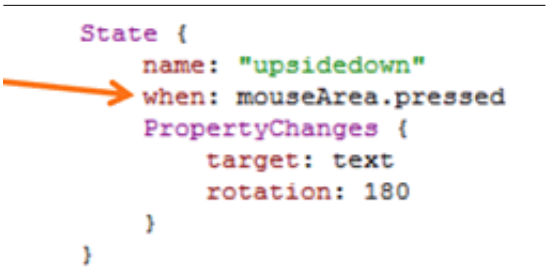
```
State {
    name: "smaller"
    extend: "shorter";
    PropertyChanges {
        target: rect
        width: 100
    }
}
```

Todos los **elementos** "Items" tienen un **estado por defecto** basado en los valores asignados a las propiedades de los elementos.

Los **elementos** pueden tener asociados distintos **estados**. Los distintos **estados** se identifican mediante un nombre excepto el **estado por defecto** que no tiene. Cada estado hereda las propiedades desde el **estado por defecto**. En el **estado** solo declaramos las diferencias entre el estado por defecto y el **estado con nombre.**

Un estado puede heredar las propiedades desde otro estado en vez de heredarlas desde el estado por defecto, **extend**

Solo puede estar activo un estado en el mismo instante de tiempo.



El **estado** puede ser activado mediante script o mediante la vinculación a eventos “**bindind**”

Transiciones entre estados (State transitions)

Las transiciones entre estados son los cambios producidos en los elementos durante el cambio de estado over at this website. Responden a la interacción de los usuarios o a eventos. Se pueden ejecutar de forma secuencial o paralela. Pueden ser animadas mediante QML. Se declaran de forma separada a los estados.

```
import Qt 4.7

Rectangle {
    // Basic hello world example
    id: root
    width: 200
    height: 200
    Text {
        id: text
        x: 66
        y: 93
        text: "Hello World"
    }

    // Declares a state where the text has been rotated 180 degrees
    // The state is entered when mouse left button is pressed
    // within mouseArea
    states: [
        State {
            name: "upsidedown"
            when: mouseArea.pressed
            PropertyChanges {
                target: text
                rotation: 180
            }
        }
    ]

    // Added a transition on top of SimpleState example
    transitions: Transition {
        NumberAnimation { property: "rotation"; duration: 500 }
    }

    // Mouse click handler, which toggles between the two states
    MouseArea {
        id: mouseArea
        anchors.fill: parent
    }

}

```

Todas las transiciones se aplican al unísono por defecto. Pueden ejecutarse en paralelo o en secuencial. Utilizamos la cláusula **form** y **to** para establecer el estado inicial y final de una transición (ámbito). Existe la posibilidad de establecer un flag (**reversible**) para animar de forma reversible las distintas transiciones secuenciales.

```
import Qt 4.7

Rectangle {
    // Basic hello world example
    id: root
    width: 200
    height: 200
    Text {
        id: text
        x: 66
        y: 93
        text: "Hello World"
    }
    // Declares a state where the text has been rotated 180 degrees
    // The state is entered when mouse left button is pressed
    // within mouseArea
    states: [
        State {
            name: "upsidedown"
            when: mouseArea.pressed
            PropertyChanges {
                target: text
                rotation: 180
            }
        }
    ]
    // Different sequential animations for press and release
    // so the speed stays constant both ways
    // Quizz:
    //  - Remove second transition and run. What happens and why?
    //  - Add reversible: true to the first transtion. What happens?
    //  - Compare original with the reversible solution.
    //    - Why's there a difference?
    //  - Remove reversible, from and to. What happens and why?
    transitions: [
        Transition {
            from: ""; to: "upsidedown"
            SequentialAnimation {
                NumberAnimation { property: "rotation"; to: 360; duration: 500 }
                NumberAnimation { property: "rotation"; duration: 250 }
            }
        },
        Transition {
            from: "upsidedown"; to: ""
            SequentialAnimation {
                NumberAnimation { property: "rotation"; to: 360; duration: 250 }
                NumberAnimation { property: "rotation"; duration: 500 }
            }
        }
    ]

    // Mouse click handler, which toggles between the two states
    MouseArea {
        id: mouseArea
        anchors.fill: parent
    }
}

```

Otro caso de interés es la posibilidad de agrupar distintas transiciones.

```
import Qt 4.7

Rectangle {
    width: 200
    height: 200

    // Add four rectangles, each default at point (25, 25)
    RedRect { id: rect1; text: "1" }
    RedRect { id: rect2; text: "2" }
    RedRect { id: rect3; text: "3" }
    RedRect { id: rect4; text: "4" }

    // Run some infinite-loop animations on the rectangles
    SequentialAnimation {
        id: anim
        running: true
        loops: Animation.Infinite
        ParallelAnimation {
            RedRectAnimation { target: rect1; x: 25; y: 25; color: "green" }
            RedRectAnimation { target: rect2; x: 150; y: 25; color: "green" }
            RedRectAnimation { target: rect3; x: 25; y: 150; color: "green" }
            RedRectAnimation { target: rect4; x: 150; y: 150; color: "green" }
        }
        PauseAnimation { duration: 1000 }
        ParallelAnimation {
            RedRectAnimation { target: rect1; x: 150; y: 25; color: "yellow" }
            RedRectAnimation { target: rect2; x: 150; y: 150; color: "yellow" }
            RedRectAnimation { target: rect3; x: 25; y: 25; color: "yellow" }
            RedRectAnimation { target: rect4; x: 25; y: 150; color: "yellow" }
        }
        PauseAnimation { duration: 1000 }
        ParallelAnimation {
            RedRectAnimation { target: rect1; x: 25; y: 25; color: "red" }
            RedRectAnimation { target: rect2; x: 25; y: 25; color: "red" }
            RedRectAnimation { target: rect3; x: 25; y: 25; color: "red" }
            RedRectAnimation { target: rect4; x: 25; y: 25; color: "red" }
        }
        PauseAnimation { duration: 1000 }
    }
}
```