Clickable, Draggable objects with QtQuick Map API

Currently you could make QML map objects clickable by including MapMouseArea for each individual map objects, however as identified by the bug report this way disables draggability of the object, thus as a workaround, normal mouse area would need to be used.

Basically with onPressed you would check whether the object is under the pressed point, and if needed mark it as pressed, and then release the pressed state in onReleased as well as with onCanceled event handlers. Then dragging could be simply handled inside the onPositionChanged events.

Finding out if an object is under the pressed point differs a bit between different objects, for Circles you could simply check whether the pressed point is closer to the center point than the radius value, for example like this:

```
var dist = Circle.center.distanceTo(map.toCoordinate(Qt.point(mouse.x,mouse.y)));
if(dist < Circle.radius){
// the mouse point is inside the circle.
}
```

Then dragging the circle could be handled simply by setting the center of the circle into the values of the mouse pointer inside the onPositionChanged handler. Anyway, you could of course take account the difference of the original center and the original pressed point to make the dragging smoother:

You could also use the onPositionChanged for resizing the circle, as illustrated in QML DynamicCircles.zip example.

With Rectangles, you would not have radius, instead you have two points. Thus with rectangle, you would need to check whether the pressed point is inside these two points. This could be done for example by utilizing following code:

```
function isPointInside(topLeft, bottomRight, point){
     if(topLeft.longitude > bottomRight.longitude){
          if(topLeft.longitude >= point.longitude && bottomRight.longitude <=
point.longitude && topLeft.latitude >= point.latitude && bottomRight.latitude <=
point.latitude){
               return true;
          }
     }else{
          if(topLeft.longitude <= point.longitude && bottomRight.longitude >=
point.longitude && topLeft.latitude >= point.latitude && bottomRight.latitude <=
point.latitude){
               return true;
          }
     }

     return false;
}
```

Then you could use the change between the original point as well as the point given in onPositionChanged to move the rectangle. Also for resizing rectangle you could use two circles marking the rectangle points as illustrated in QML DynamicRectangle.zip example.

With MapImage you actually get only one coordinate point which is the left upper corner. Then you also know that the image will not be scaled, thus if you know the pixel size of the image, you could indeed calculate whether the given point clicked by mouse would be inside the marker image area on the screen. This can be calculated for example with following code snipped:

```
function selectMarkerIcon(mx, my){
     for (var i = 0; i < icons.objects.length; ++i) {
          var topLeftPoint = map.toScreenPosition(icons.objects[i].coordinate);
          var xStart = parseInt(topLeftPoint.x);
```

```
            var yStart =  parseInt(topLeftPoint.y);

            if((mx >= xStart) && (my >= yStart) && (mx <= (xStart + xwitdh)) && (my <=
(yStart + xhight))){
                 return i;
            }
       }
       return -1;
}
```

In this example MapImages are expected to be stored in a MapGroup called icons, and if a more dynamic way is required a JavaScript array method could be used. Full example for Draggable MapImages is shown in QML MapMarker.zip example.