Dokumentation der Implementierung der Smells im Aktivitätsdiagramm

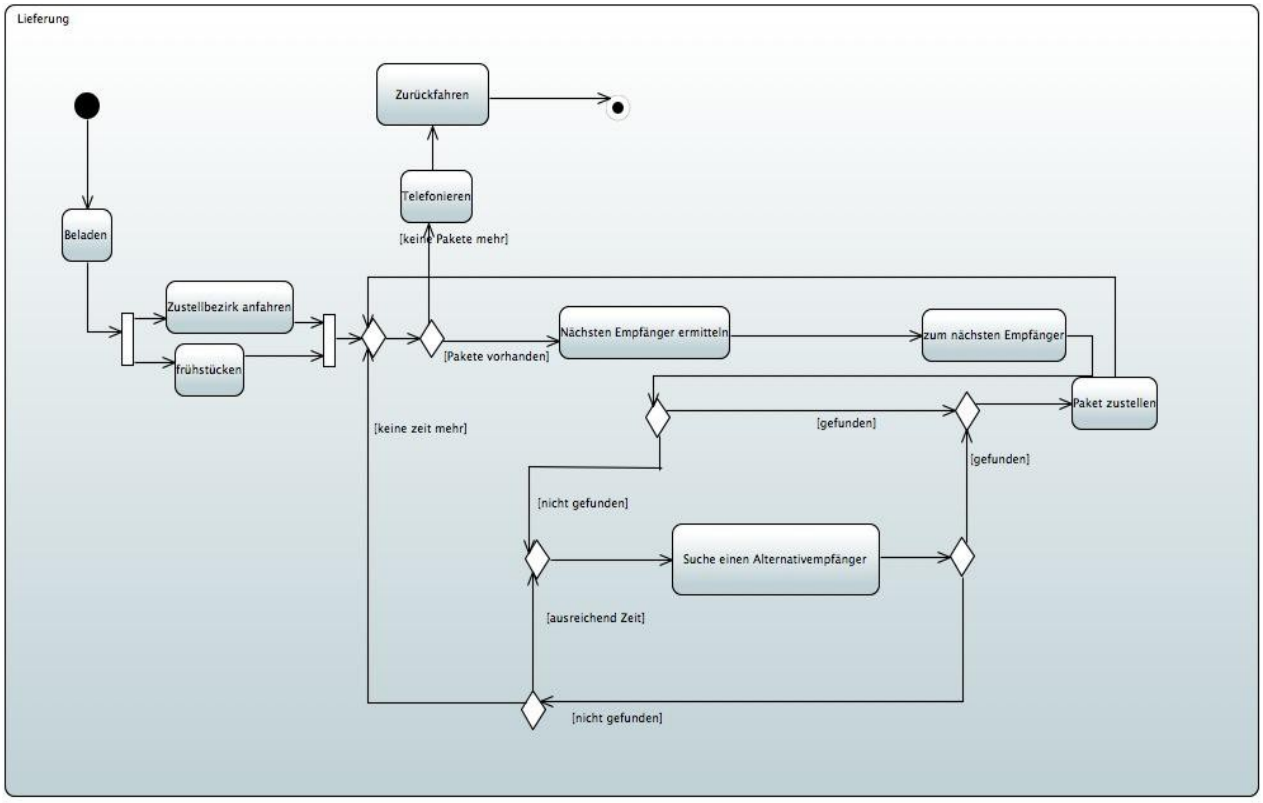


Bild 1

Für das folgende Aktivitätsdiagramm sollen die konfigurierten Smells 'Merge Node connected with a Decision Node by a Transition' sowie 'Opaqua Action Node concected with Decision Node by a Transition ' ermittelt werden. Erwartet werden also folgende im Bild dargstellten Bereiche.

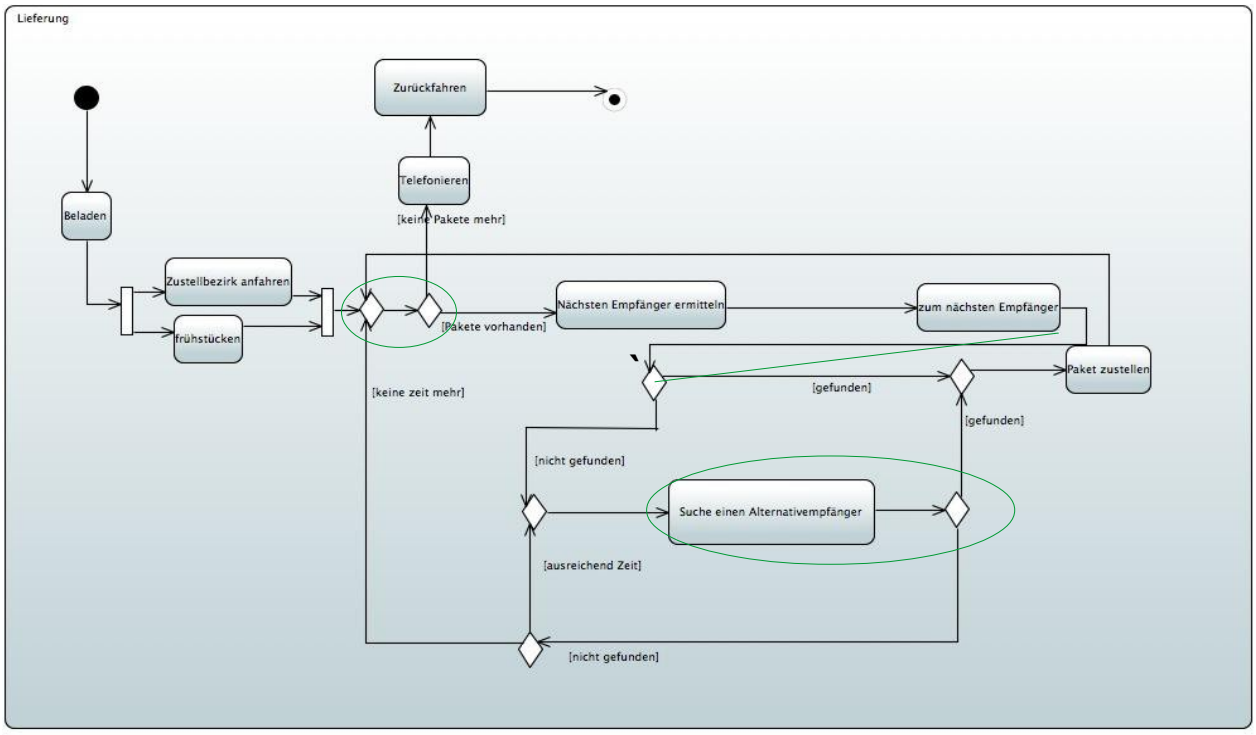


Bild 2

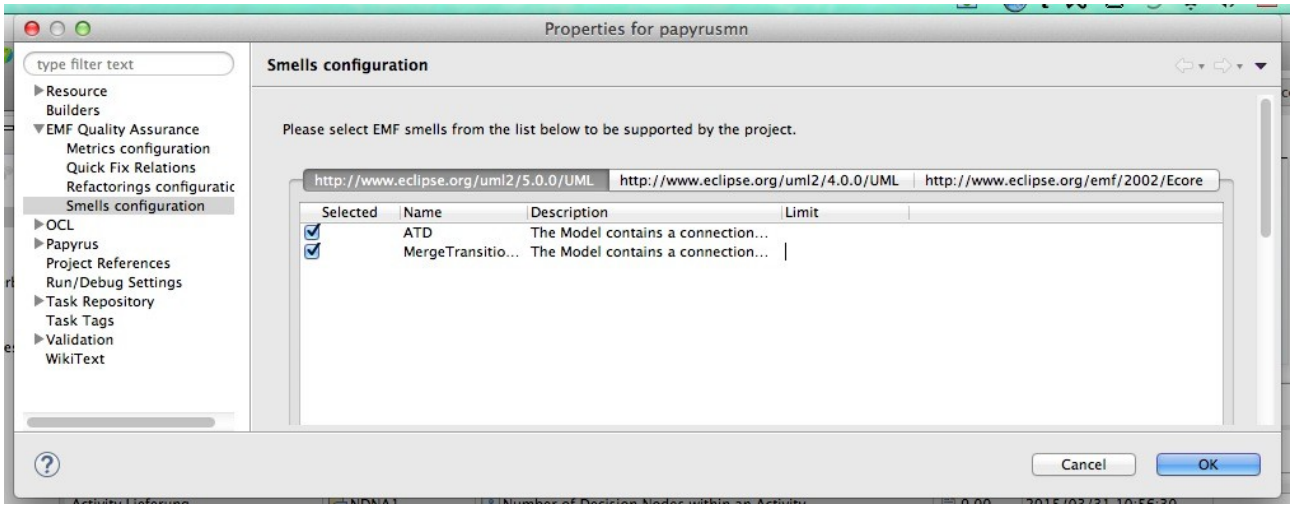


Bild 3

Hier sieht man soweit die angewählten Smells.

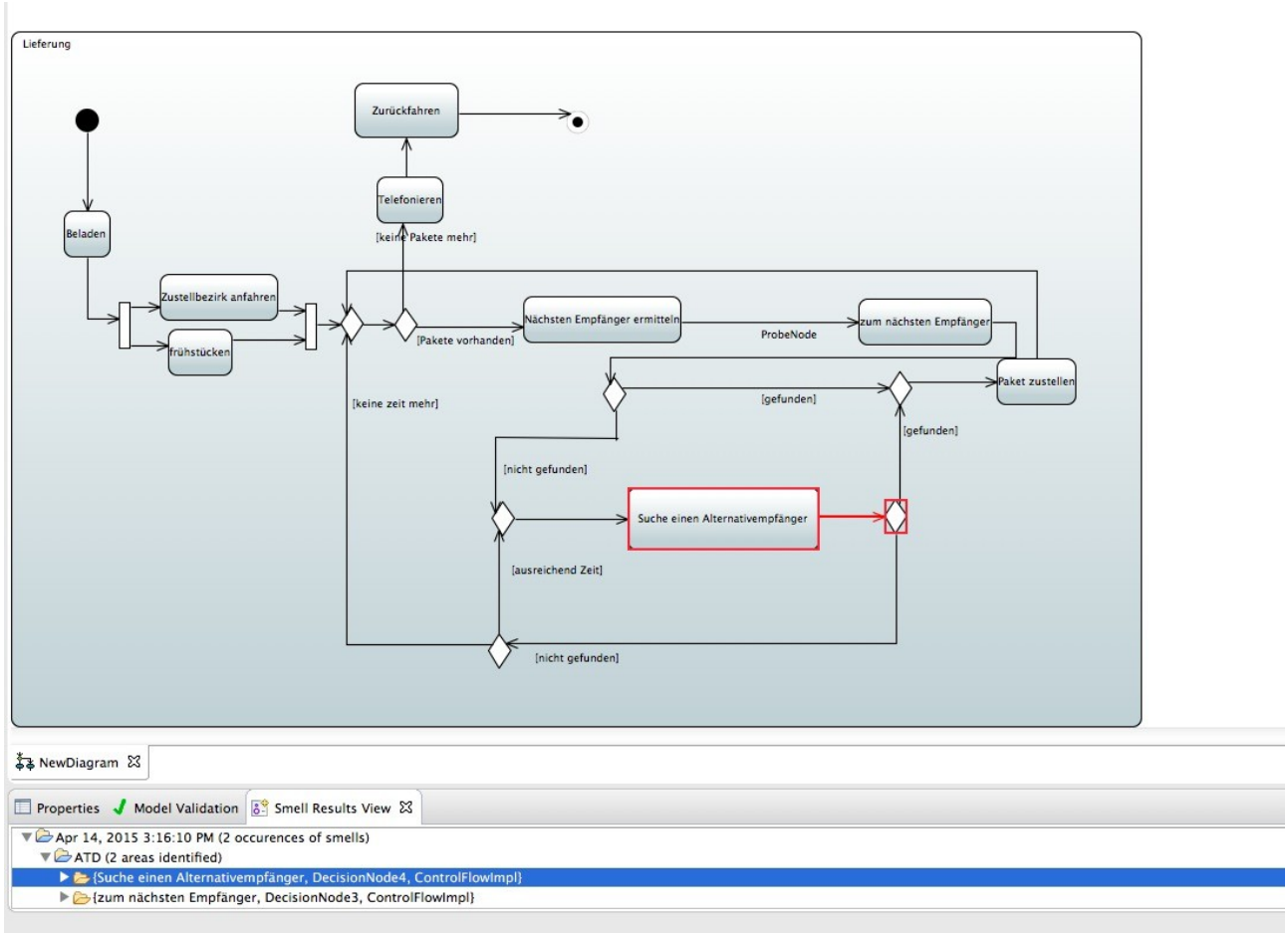
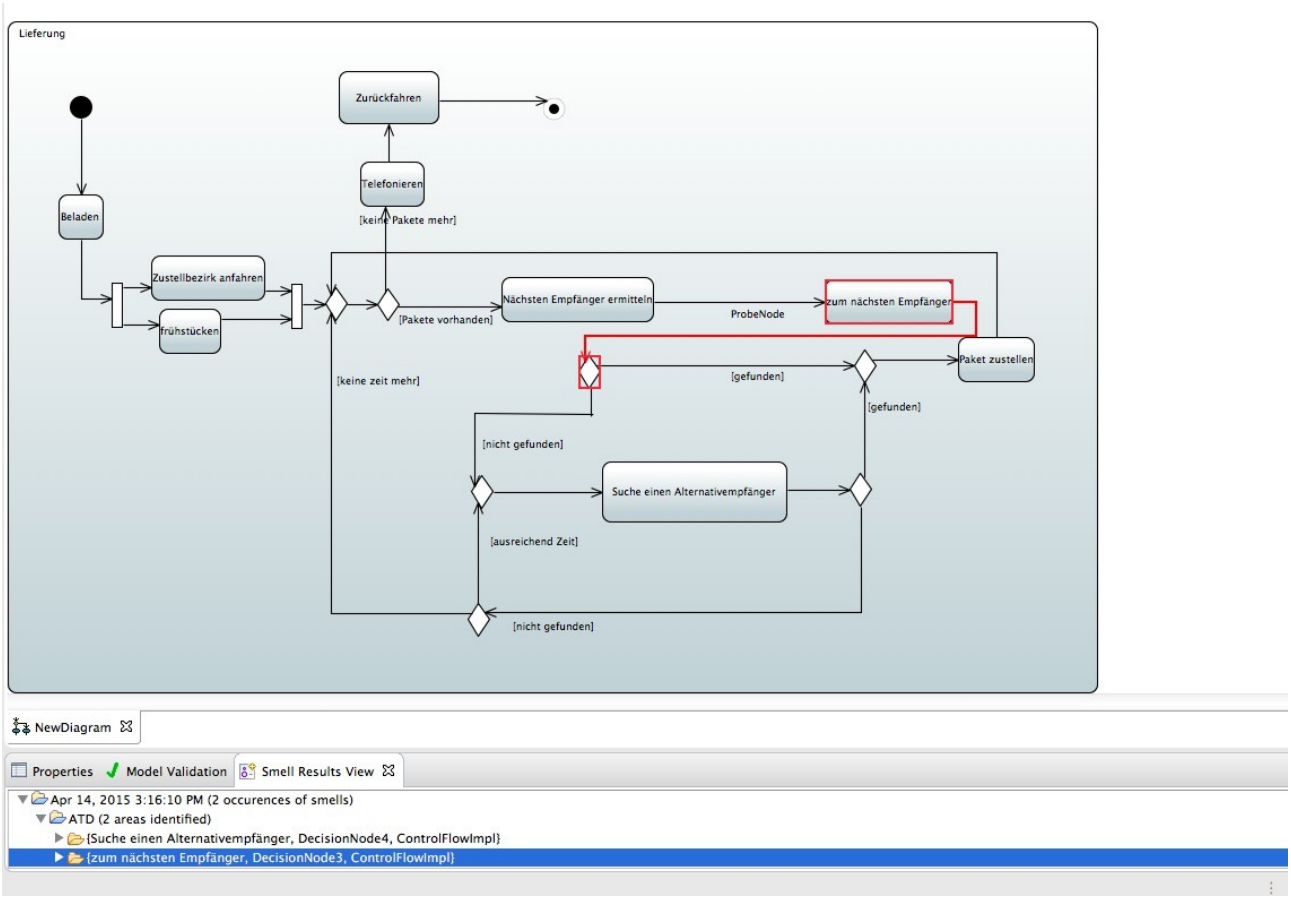


Bild 4

Einer der beiden 'Opaqua Action Node concected with Decision Node by a Transition '.

Bild 5

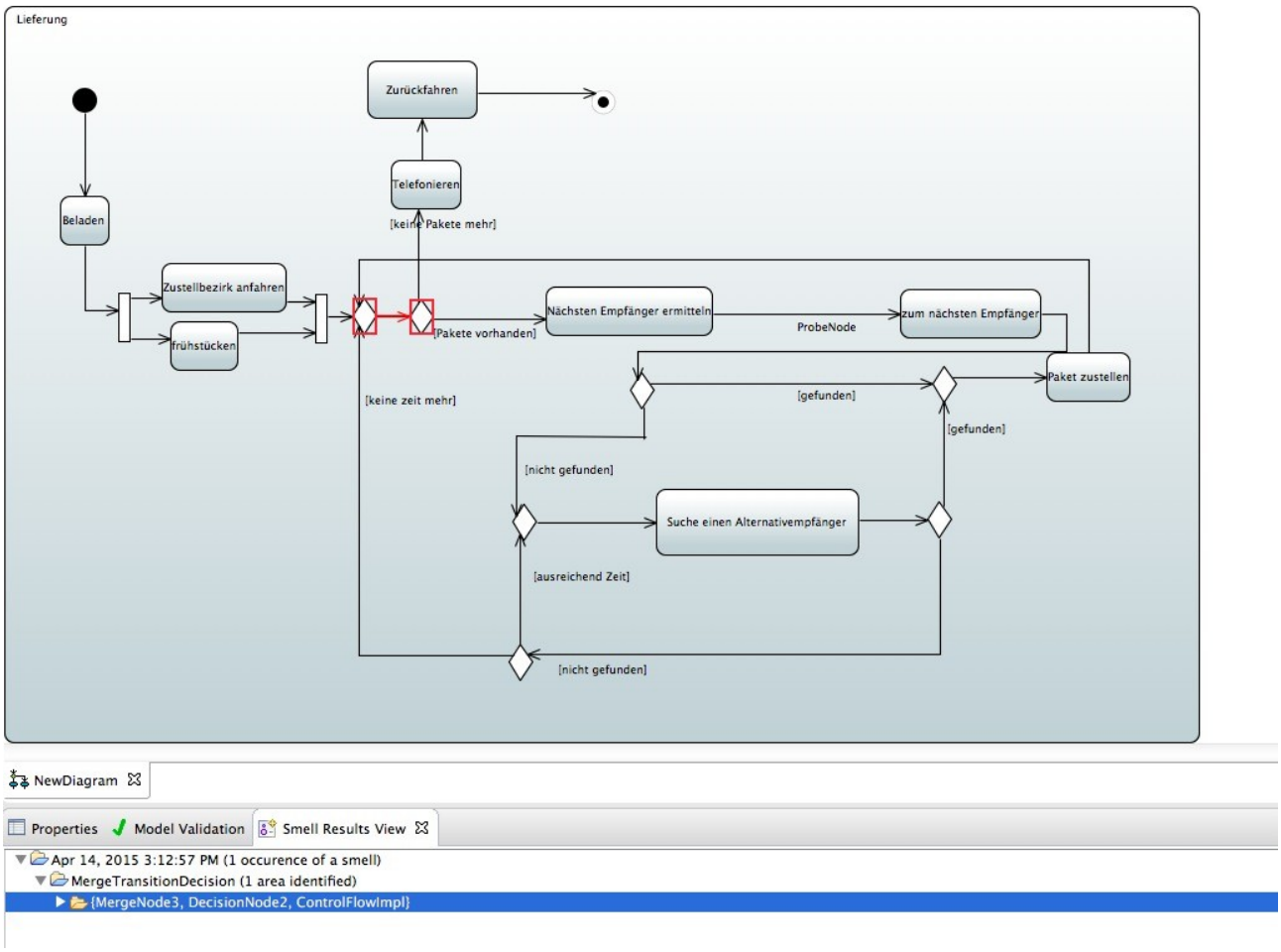


Beide 'Opaqua Action Node concected with Decision Node by a Transition '. (gelegentlich werden beide alle auftretenden Smells gleichzeitig angezeigt).

Zur Implementierung der jeweiligen Smells:

Bild 6

Der 'Merge Node connected with a Decision Node by a Transition' Smell. Zusammengefasst wurden alle Smells, welche man angezeigt haben wollte, auch soweit angezeigt.



Zur Implementierung der jeweiligen Smells:

MergeTransitionDecision - The Model contains a connection between a Merge- and a DecisionNode

```
	public LinkedList<LinkedList<EObject>> findSmell(EObject root) {
		LinkedList<LinkedList<EObject>> results = new
LinkedList<LinkedList<EObject>>();
		Activity model = (Activity) root;
			EList<ActivityEdge> allActivityEdges =
getAllActivityEdges(model);
			for (ActivityEdge activityEdge : allActivityEdges) {
				if ( (activityEdge.getSource() instanceof MergeNode)
						&& (activityEdge.getTarget() instanceof
DecisionNode) ) {
					LinkedList<EObject> result = new
LinkedList<EObject>();
					result.add(activityEdge.getSource());
					result.add(activityEdge.getTarget());
```

```
                        result.add(activityEdge);
                        // TODO
                        results.add(result);
                    }
                }
            return results;
        }
```

Die Methode findSmell hat den Rückgabetyp LinkedList<LinkedList<EObject>> und nimmt als Paramater ein EObject root. Die übergeordnete Liste steht für die nach Smells auszutestende Aktivität. Ihre Listen selbst sind nun jeweilige Smells. Der übergebene Paramter ist hierbei die Aktivität. Von der Idee her untersucht der Algorithmus nun die ActivityEdges nach Smells. Dafür wird eine Liste allActivityEdges erstellt und diese dann durch die Methode getAllactivityEdges(model) mit allen ActivityEdges aus der Aktivität gefüllt. In einer for-Schleife wird nun mit der Laufvariablen activityEdge vom Typ ActivityEdge über die Liste mit allen ActivityEdges iteriert. Wenn die if-Abfrage bei einem activityNode festellt, dass der Sourceknoten wie der Targetknoten vom Typ MergeNode und DecisionNode sind, geht es weiter im ifRumpf. Hier wird eine neue Liste result erstellt und in ihr werden nun der DecisionNode, die verbindende ActivityEdge sowie der Mergenode abgelegt. Diese Liste result wird nun in der übergeordneten Liste results abgelegt und nach Ablauf aller Iterationen wird results über return zurückgegeben.

ATD – The Model contains a connection between a Action- and a DecisionNode

```
      public LinkedList<LinkedList<EObject>> findSmell(EObject root) {
            LinkedList<LinkedList<EObject>> results = new
LinkedList<LinkedList<EObject>>();
            Activity model = (Activity) root;
                  EList<ActivityEdge> allActivityEdges =
getAllActivityEdges(model);
                  for (ActivityEdge activityEdge : allActivityEdges) {
                        if ( (activityEdge.getSource() instanceof OpaqueAction)
                                    && (activityEdge.getTarget() instanceof
DecisionNode) ) {
                              LinkedList<EObject> result = new
LinkedList<EObject>();
                              result.add(activityEdge.getSource());
                              result.add(activityEdge.getTarget());
                              result.add(activityEdge);
                              // TODO
                              results.add(result);
```

```
                    }
                }
            return results;
        }
```

Der Code arbeitet genauso wie bei MergeTransitionDecision mit dem einzigen Unterschied, dass hier mit if nach anderen zusammenhängenden Knoten gesucht wird. Also in if wird nun nach ActivityEdges gesucht, die als Source nicht mehr MergeNode, sondern einen ActivityNode vom Typ OpaqueAction haben. Ansonsten ist es vom Schema her genau dasselbe.
Ausgegeben wird wiederum results als Liste.