Dokumentation der Implementierung der Metriken im Aktivitätsdiagramm

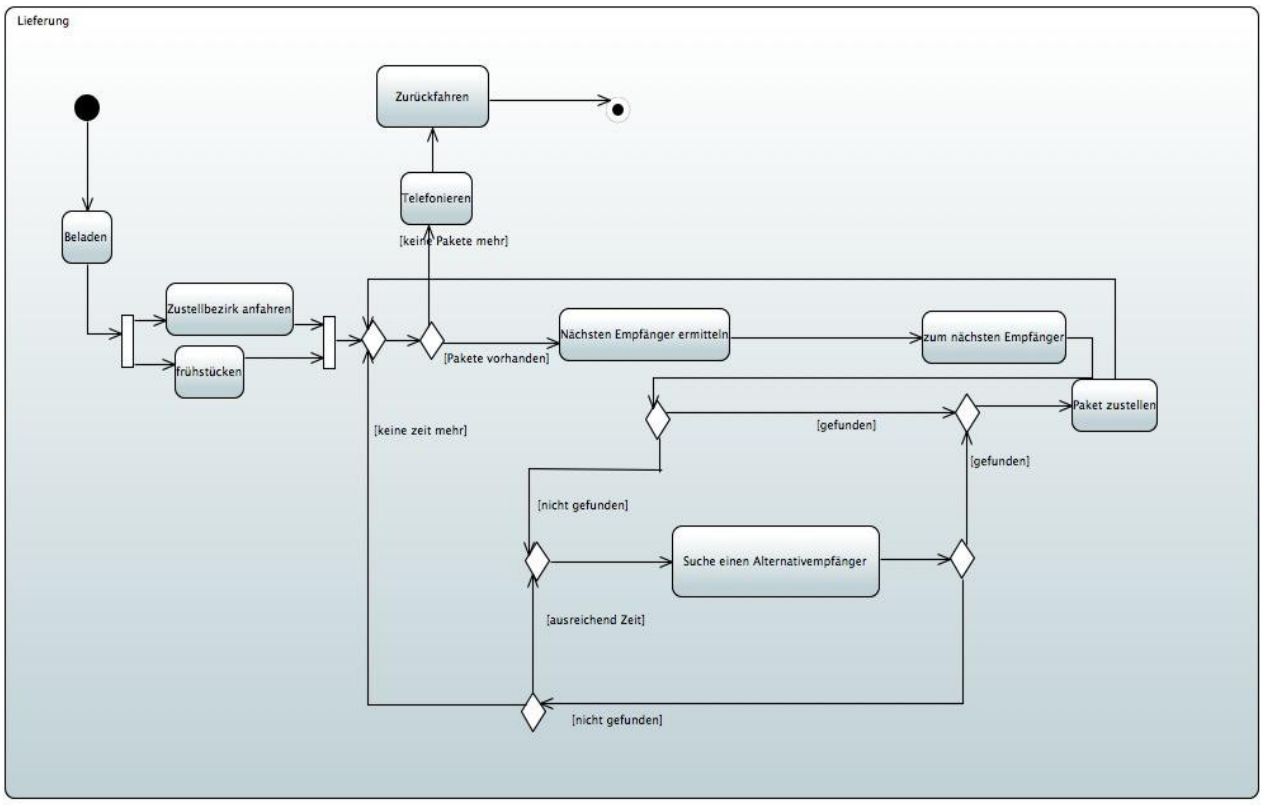


Bild 1

Im vorliegenden Aktivitätsdiagramm (Bild 1) befinden sich insgesamt wie sich nach einfachem Nachzählen ergeben hat, 20 Knoten und 24 Tranisitionen. Die Knoten teilen sich auf in 1 Forknoten 1 Joinknoten, 1 Initialknoten, 1 Finite Knoten, 9 Aktionsknoten, 4 Decisionknoten und 3 Mergeknoten, weiter gibt es noch 24 Transionen. Die Auswahl der implementierten Metriken zur Berechnung der Anzahl der genannten Diagrammtypen wurde im folgenden Bild 2 getroffen.

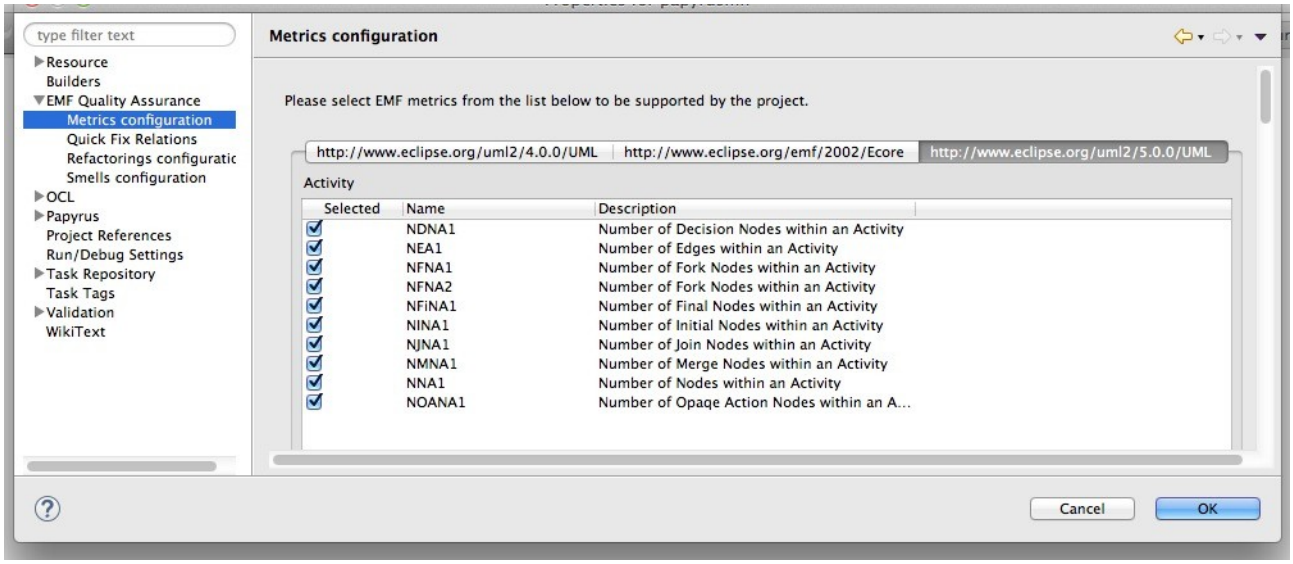
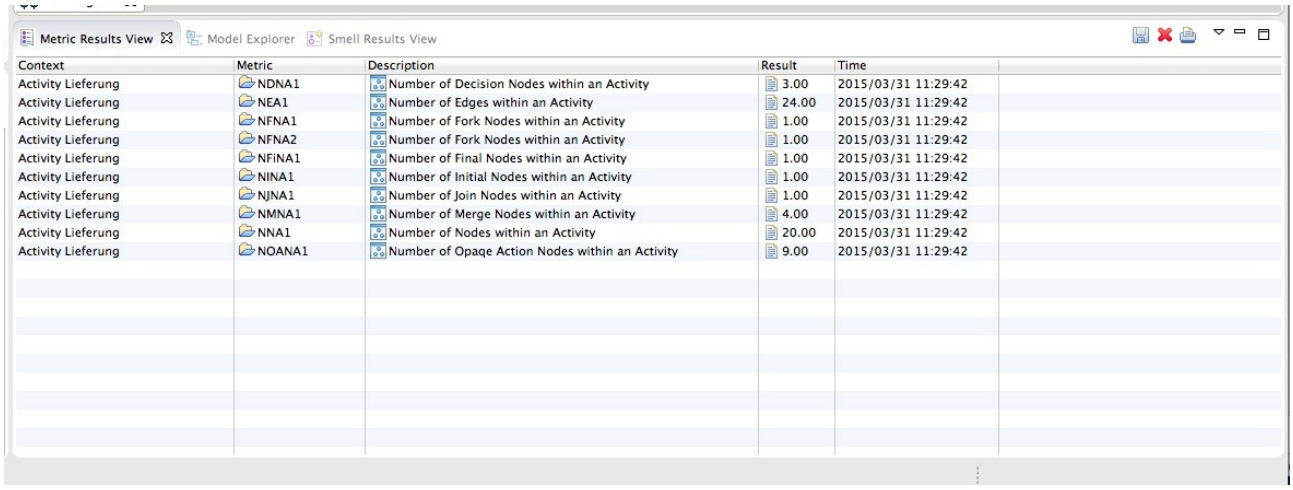


Bild 2

Das Ergebnis sieht man im folgenden Screenshot der sich ergebenden “Metric Result View” (siehe

Bild 3)



Context	Metric	Description	Result	Time
Activity Lieferung	NDNA1	Number of Decision Nodes within an Activity	3.00	2015/03/31 11:29:42
Activity Lieferung	NEA1	Number of Edges within an Activity	24.00	2015/03/31 11:29:42
Activity Lieferung	NFNA1	Number of Fork Nodes within an Activity	1.00	2015/03/31 11:29:42
Activity Lieferung	NFNA2	Number of Fork Nodes within an Activity	1.00	2015/03/31 11:29:42
Activity Lieferung	NFiNA1	Number of Final Nodes within an Activity	1.00	2015/03/31 11:29:42
Activity Lieferung	NINA1	Number of Initial Nodes within an Activity	1.00	2015/03/31 11:29:42
Activity Lieferung	NJNA1	Number of Join Nodes within an Activity	1.00	2015/03/31 11:29:42
Activity Lieferung	NMNA1	Number of Merge Nodes within an Activity	4.00	2015/03/31 11:29:42
Activity Lieferung	NNA1	Number of Nodes within an Activity	20.00	2015/03/31 11:29:42
Activity Lieferung	NOANA1	Number of Opaqe Action Nodes within an Activity	9.00	2015/03/31 11:29:42

Bild 3

Wie man sieht, sind die abgezählten Werte genau dieselben wie die berechneten.

Anmerkung: Die Decisiosnodes wurden 2 Mal berechnet (NDNA1 und NDNA)

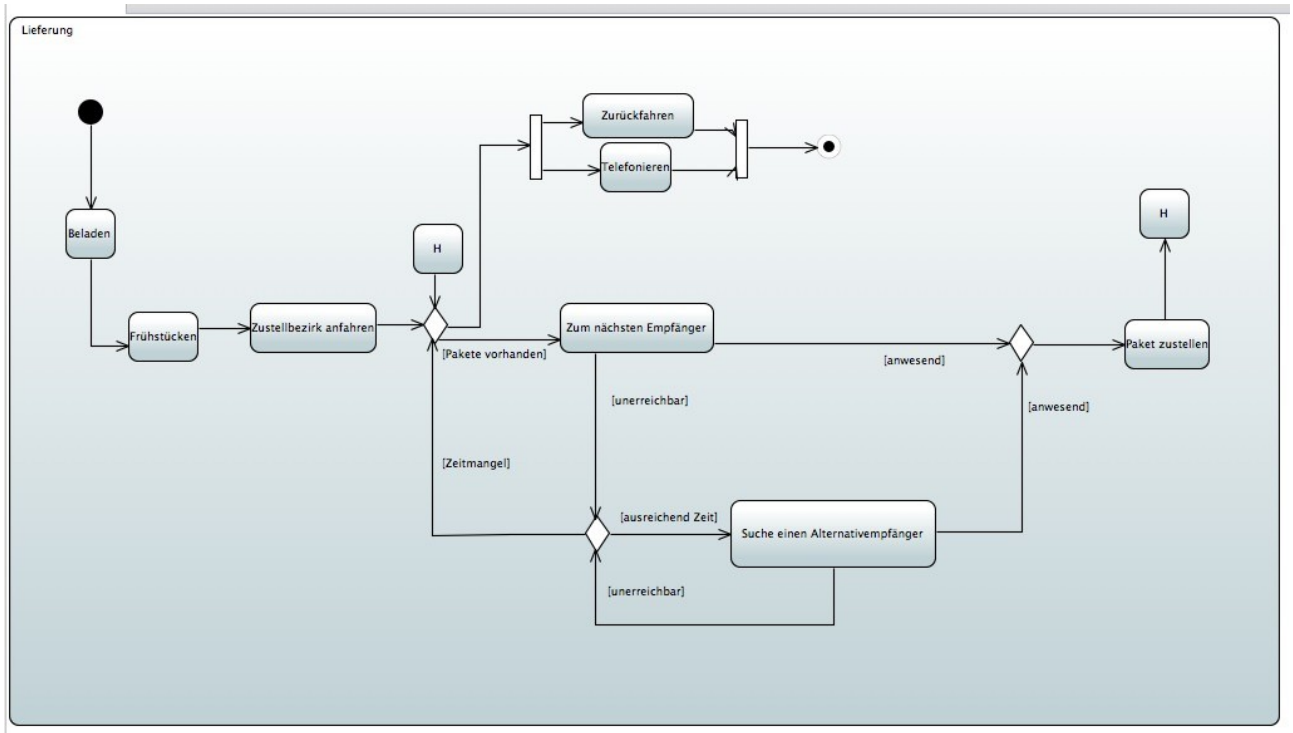
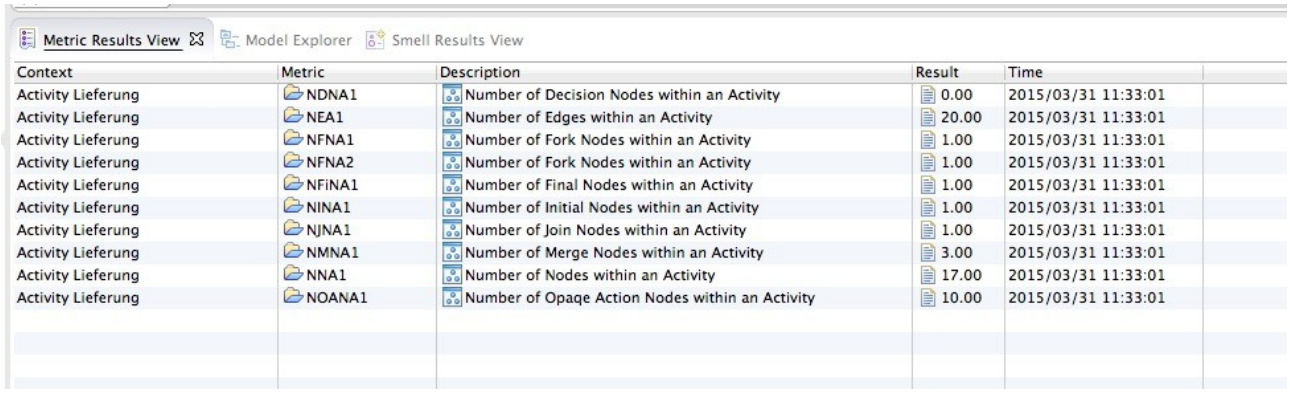


Bild 4

Dieses Bild stellt das Augangsdiagramm (Bild 1) nach Anwendung einiger der Techniken dar. Es besteht nach Nachzählung insgesamt aus 17 Knoten und 20 Transitionen. Von den Knoten sind es 1 Forkknoten, 1 Join Knoten, 1 Initialknoten, 1 Finite Knoten, 10 Aktionsknoten und 3 Mergeknoten.

An der Auswahl hat sich nichts zu Bild 2 geändert.

Folgendes Bild ergibt sich nun.



Metric Results View | Model Explorer | Smell Results View

Context	Metric	Description	Result	Time
Activity Lieferung	NDNA1	Number of Decision Nodes within an Activity	0.00	2015/03/31 11:33:01
Activity Lieferung	NEA1	Number of Edges within an Activity	20.00	2015/03/31 11:33:01
Activity Lieferung	NFNA1	Number of Fork Nodes within an Activity	1.00	2015/03/31 11:33:01
Activity Lieferung	NFNA2	Number of Fork Nodes within an Activity	1.00	2015/03/31 11:33:01
Activity Lieferung	NFiNA1	Number of Final Nodes within an Activity	1.00	2015/03/31 11:33:01
Activity Lieferung	NINA1	Number of Initial Nodes within an Activity	1.00	2015/03/31 11:33:01
Activity Lieferung	NJNA1	Number of Join Nodes within an Activity	1.00	2015/03/31 11:33:01
Activity Lieferung	NMNA1	Number of Merge Nodes within an Activity	3.00	2015/03/31 11:33:01
Activity Lieferung	NNA1	Number of Nodes within an Activity	17.00	2015/03/31 11:33:01
Activity Lieferung	NOANA1	Number of Opaqe Action Nodes within an Activity	10.00	2015/03/31 11:33:01

Bild 5

Abermals haben alle berechneten Metrike dieselben Werte wie die von Hand gezählten.

Eine Übersicht der einzelnen Implementierungen ergibt folgendes Bild:

Die Methode zur Berechnung der jeweiligen Diagrammtypen mit java ist jeweils die Folgende

```
public double calculate() {
	org.eclipse.uml2.uml.Activity in = (org.eclipse.uml2.uml.Activity) context.get(0);
	double ret = 0.0;
	// begin custom code

	// end custom code
	return ret;
```

Die Methode calculate weist der Variablen in die Aktivität mit den abzuzählenden Elementen zu und initialisiert einen Doublewert ret. Nun folgt der das metrikszpefische Codestück zur Berechnung der jeweiligen Metrik, zwischen // begin custom code und // end custom code, was in den nächsten Zeilen jeweils erläutert wird. Nach der Berechnung dieser Metrik und dem Festehalten des Wertes in der Variablen ret wird diese zum Schluss ausgegeben.

NFNA1 – Number of Fork Nodes within an Acitivty

```
	org.eclipse.uml2.uml.Activity in = (org.eclipse.uml2.uml.Activity)
context.get(0);
	double ret = 0.0;
	// begin custom code
	for (ActivityNode node : in.getNodes()) {
		if (node instanceof ForkNode) {
			ret++;
		}
	}
	// end custom code
```

In einer For-Schleife wird mit der Variablen node vom Typ ActivityNode der Inhalt von in.getNodes() durchlaufen. Je Schleifendurchlauf der Typ von node mit dem ForkNodeTyp mit if verglichen, falls ja, wird der Wert von ret um 1 erhöht.

NDNA1 – Number of Decision Nodes within an Activity

```
// begin custom code
for (ActivityNode node : in.getNodes()) {
	if (node instanceof DecisionNode) {
		ret++;
	}
}
// end custom code
```

In einer For-Schleife wird mit der Variablen node vom Typ ActivityNode der Inhalt von in.getNodes() durchlaufen. Je Schleifendurchlauf der Typ von node mit dem DecisionNodeTyp mit if verglichen, falls ja, wird der Wert von ret um 1 erhöht.

NJNA1 – Number of Join Nodes within an Activity

```
// begin custom code
for (ActivityNode node : in.getNodes()) {
	if (node instanceof JoinNode) {
		ret++;
	}
}
// end custom code
```

In einer For-Schleife wird mit der Variablen node vom Typ ActivityNode der Inhalt von in.getNodes() durchlaufen. Je Schleifendurchlauf der Typ von node mit dem JoinNodeTyp mit if verglichen, falls ja, wird der Wert von ret um 1 erhöht.

NINA1 – Number of Initial Nodes within an Activity

```
// begin custom code
for (ActivityNode node : in.getNodes()) {
	if (node instanceof InitialNode) {
		ret++;
	}
}
// end custom code
```

In einer For-Schleife wird mit der Variablen node vom Typ ActivityNode der Inhalt von in.getNodes() durchlaufen. Je Schleifendurchlauf der Typ von node mit dem InitialNodeTyp mit if verglichen, falls ja, wird der Wert von ret um 1 erhöht.

NFINA – Number of Final Nodes within an Activity

```
// begin custom code
for (ActivityNode node : in.getNodes()) {
	if (node instanceof FinalNode) {
		ret++;
	}
}
// end custom code
```

In einer For-Schleife wird mit der Variablen node vom Typ ActivityNode der Inhalt von in.getNodes() durchlaufen. Je Schleifendurchlauf der Typ von node mit dem FinalNodeTyp mit if verglichen, falls ja, wird der Wert von ret um 1 erhöht.

NNA1 – Number of Nodes within an Activity

```
double ret = 0.0;
// begin custom code
for (ActivityNode node : in.getNodes()) {
	{
		ret++;
	}
}
// end custom code
```

In einer For-Schleife wird mit der Variablen node vom Typ ActivityNode der Inhalt von in.getNodes() durchlaufen. Je Schleifendurchlauf wird der Wert von ret um 1 erhöht.

NEA1 – Number of Edges within an Activity

```
// begin custom code
for (ActivityEdge edge : in.getEdges()) {
	{
		ret++;
	}
}
// end custom code
```

In einer For-Schleife wird mit der Variablen edge vom Typ ActivityEdge der Inhalt von in.getNodes() durchlaufen. Je Schleifendurchlauf wird der Wert von ret um 1 erhöht.

NOANA1 – Number of Opaque Action Nodes within an Activity

```
// begin custom code
for (ActivityNode node : in.getNodes()) {
      if (node instanceof OpaqueAction) {
            ret++;
      }
}
// end custom code
```

In einer For-Schleife wird mit der Variablen node vom Typ ActivityNode der Inhalt von in.getNodes() durchlaufen. Je Schleifendurchlauf der Typ von node mit dem OpaqueActionTyp mit if verglichen, falls ja, wird der Wert von ret um 1 erhöht.

NMNA1 – Number of Merge Nodes within an Activity

```
// begin custom code
for(ActivityNode activity : in.getNodes()) {
      if(activity instanceof MergeNode){
            ret++;
      }
}
// end custom code
```

In einer For-Schleife wird mit der Variablen node vom Typ ActivityNode der Inhalt von in.getNodes() durchlaufen. Je Schleifendurchlauf der Typ von node mit dem MergeNodeTyp mit if verglichen, falls ja, wird der Wert von ret um 1 erhöht.

NFNA2 – Number of Fork Nodes within an Acitivty

```
	private final String expression =
		"self.node -> select(oclIsTypeOf(ForkNode)) -> size()";
	private List<EObject> context;

	@Override
	public void setContext(List<EObject> context) {
		this.context = context;
	}

	@Override
	public double calculate() {
		EObject contextObject = context.get(0);
		return OCLManager.evaluateOCLOnContextObject(contextObject,
expression);
```

Wie bei NFNA1 geht es hier wieder um die ForkNodes, mit dem Unterschied, dass die Implementierung nicht in Java, sondern in OCL erfolgt ist.
Auch hier wird von der Methode calculate der metrische Wert zurückgeben. Doch die Berechnung erfolgt über die Klassenmethode OCLManager.*evaluateOCLOnContextObject*(contextObject, expression).
Wichtig ist der Aufbau der als Paramter übergebenen Stringvariable expression. Übergeben wird wieder die gesamte Aktivität als self mit den zu zählen Elementen, hier sind das die ForkNodes. Nun wird jeder einzelne Knoten übergeben und im Falle einer Gleichheit mit dem ForkNodetyp markiert. Schliesslich berechnet size() noch den Wert aller markierten Nodes, was dann auch gleichzeitig die Anzahl aller ForkNodes ist.