YAMM

YAMM ist eine Objektorientierte (Meta-) Modellier- und Simulationsumgebung. Der Kern des Systems ist eine Programmier- bzw. Modelliersprache, die grafische und textuelle Elemente kombiniert. Sie orientiert sich am klassischen Objektorientierten Programmierkonzept. Die Klassenstruktur und Kontrollstrukturen werden grafisch modelliert (Klassendiagram und Zustandsdiagram). In das Zustandsdiagramm ist eine Anweisungssprache (Expressionlanguage) integriert, die wie herkömmliche Programmiersprachen aufgebaut ist (Java-Bibliotheken können integriert werden). Zusätzlich ist ein Discrete-Event-System enthalten, bei dem der Programmablauf über das Senden und Empfangen von Events gesteuert wird.

Zur Untersuchung der modellierten Programme ist ein Simulations-/Debugsystem enthalten mit dem die Programme schrittweise durchlaufen werden können. Die erzeugten Objekte werden in einem Objektdiagramm angezeigt und können editiert werden.

YAMM lässt sich mit animationsfähigen grafischen Notationen für die erstellten Modelle erweitern (ausführbare Modellierungssprache). Die Ausführungssemantik wird dabei im YAMM-Modell angegeben und steuert die Animation im erzeugten Editor.

Mit einem Discrete-Event-System werden über Events gesteuerte Modelle erstellt, und untersucht.

1 Klassendiagramm

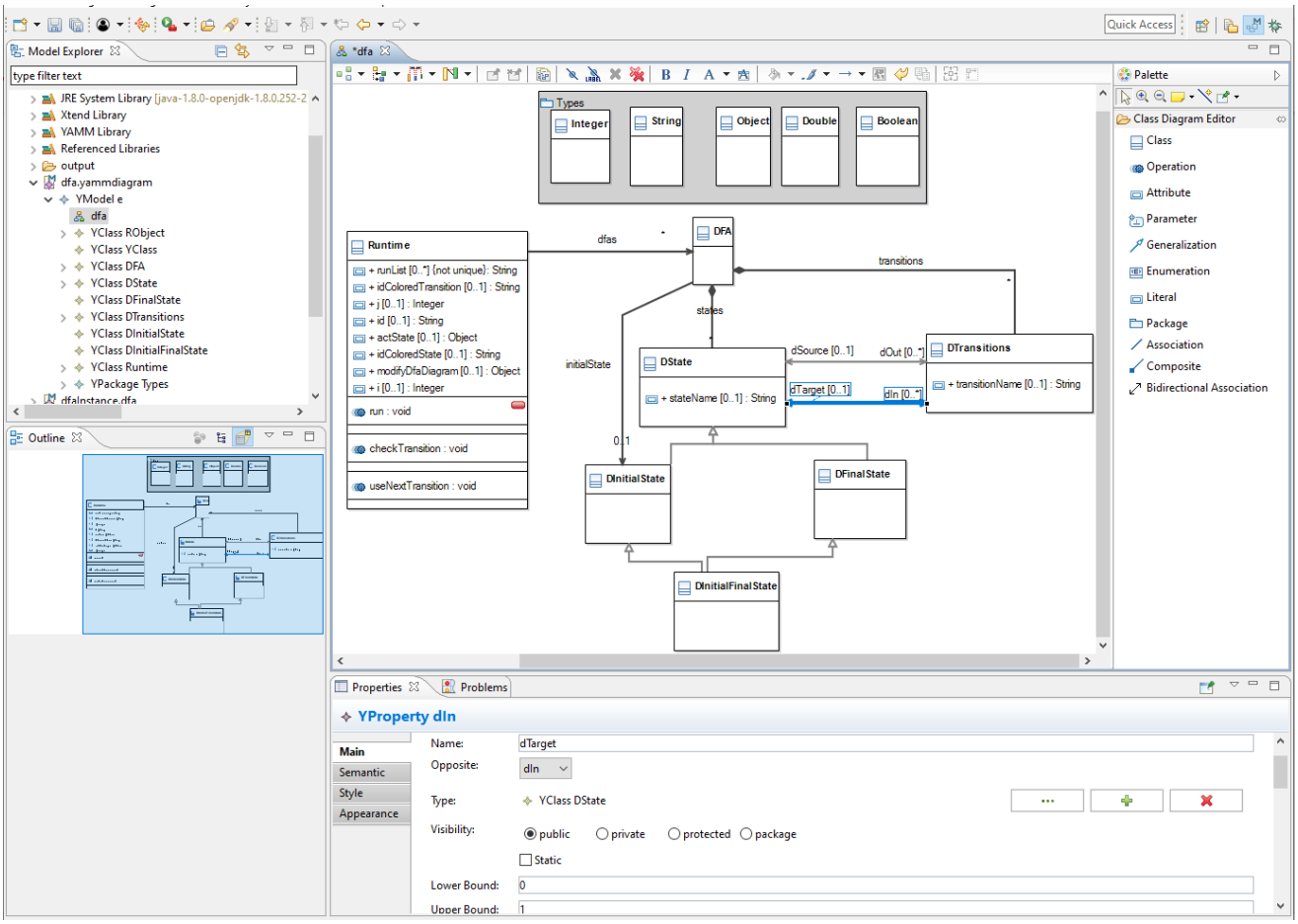

Abbildung 1: YAMM-Klassendiagramm

Zum Erstellen der YAMM-Modelle ist ein Klassendiagram-Editor enthalten. Die Syntax orientiert sich am UML-Klassendiagram. Alle verfügbaren Elemente sind in der Palette auswählbar und werden mit der Maus auf dem Diagramm erstellt. In der Properties-View werden die Eigenschaften der im Diagramm markierten Elemente angegeben.

Das Programm startet bei der als initial-aktiv markierten Operation (es können mehrere angegeben werden).

Zusätzlich gibt es eine Funktion zum automatischen Layouten der Diagramme und es können Notizen eingefügt werden.

Der Editor ist als Eclipse-Plugin mit Eclipse-Sirius entwickelt.

2 Zustandsdiagramm und Expressionlanguage

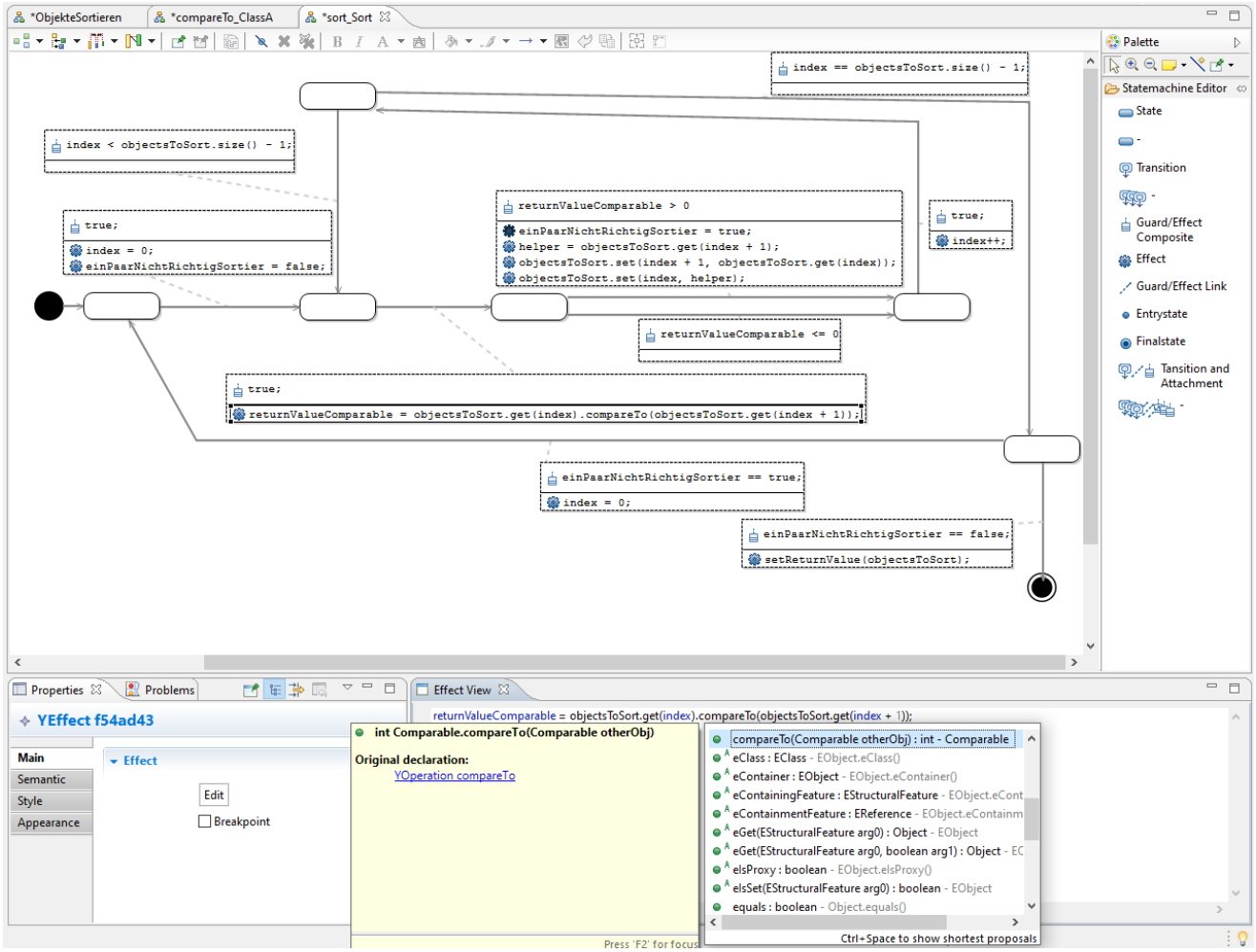

Abbildung 2: YAMM-Zustandsdiagramm mit Eingabe der Expressionlanguage

Das Verhalten der Operationen wird als Zustandsdiagram, in Verbindung mit einer Expressionlanguage definiert. Der Editor wird mit einem Doppelklick auf eine Operation im Klassendiagram erstellt bzw. geöffnet.

Die Ausführung eines Zustandsdiagramms startet beim Entrystate und durchläuft das Model bis zum Finalstate. Bei jedem Zustand werden die Guards aller ausgehenden auf ihren Wahrheitswert geprüft. Diejenige Transition, dessen Guard wahr ist wird geschaltet und die zugehörigen Effekte werden aufgeführt.

Für die Guards und Effekte wird eine Expressionlanguage verwendet, in der Java-Ausdrücke verwendet werden. Bei der Eingabe wird der Benutzer durch ein Assistenz Systeme mit automatischer Vervollständigung und Syntaxhervorhebung unterstützt. In der Sprache können alle Java-Bibliotheken verwendet werden. Die Java Standard-Bibliotheken sind standartmäßig integriert.

3 Fehleranzeige/Validierung

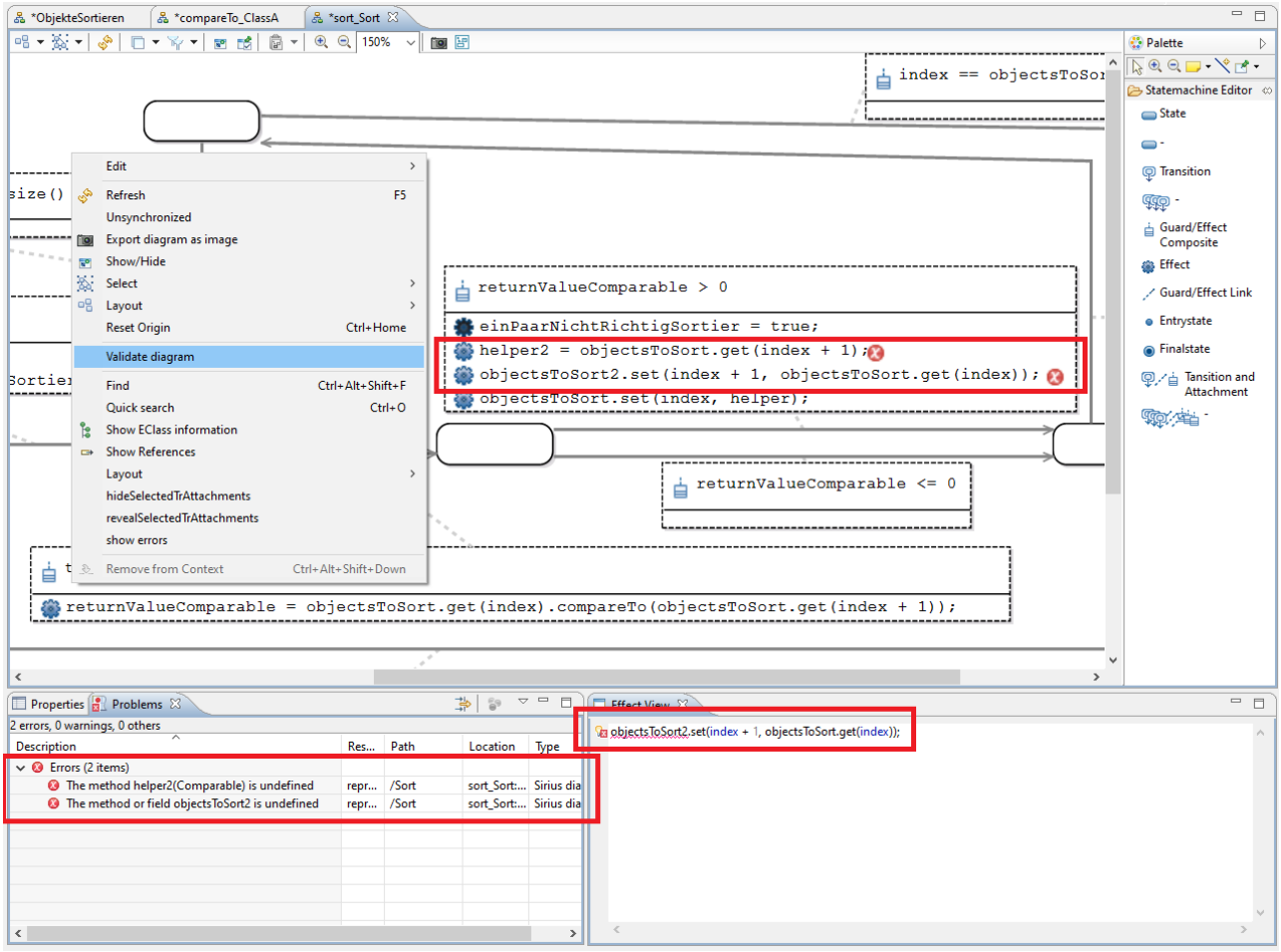

Abbildung 3: Validierung von YAMM-Modellen

Die Diagramme können mit einem Validierungssystem auf Fehler überprüft werden. Im Zustandsdiagram sind dies Syntaktische Fehler in der Expressionlanguage und im Klassendiagram Doppelte Namen und nicht gesetzte Typen von Attributen. Mit einem Doppelklick auf die Anzeigen in der Problems-View wird zu den entsprechenden Anzeigen im Diagramm navigiert.

4 Debugsystem

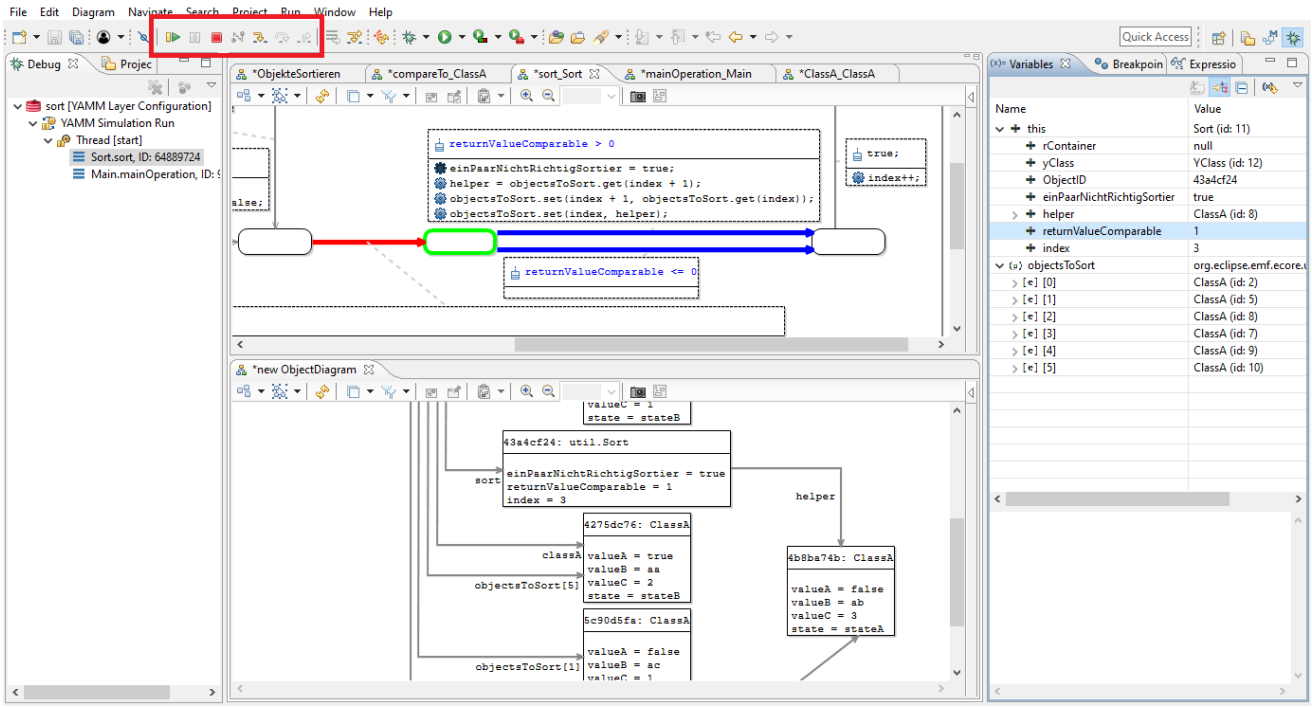

Abbildung 4: YAMM-Debugger

Mit dem Debugger können YAMM-Modelle ausgeführt und untersucht werden. Mit den im Bild markierten Schaltflächen wird die Simulation gesteuert. Sie kann schrittweise durchlaufen werden, oder bis zum nächsten mit einem Breakpoint markierten Effekt durchlaufen werden. Weiterhin besteht der Debugger aus drei Teilen:

1. Das Zustandsdiagram der aktuell aktiven Operation wird automatisch geöffnet und der Fortschritt im Diagramm angezeigt. Der aktive Zustand und die im Nächsten Simulationsschritt auszuführenden Guards oder Effekte werden farbig markiert.

2. Die Werte der Attribute des Objektes zu dem die aktive Operation gehört, sowie die Werte der Parameter der Operation werden in der Variables-View angezeigt. In der Debug-View wird der Operationsstapel und die erzeugten Threads angezeigt.

3. Zusätzlich werden die erzeugten Objekte im Objektediagram angezeigt. Die Werte der Attribute können während der Simulation verändert werden und es können Objekte erzeugt und gelöscht werden. Mit dem Objektdiagramm kann auch vor dem Simulationsstart ein Initialzustand erzeugt werden.

5 Layer-Editoren

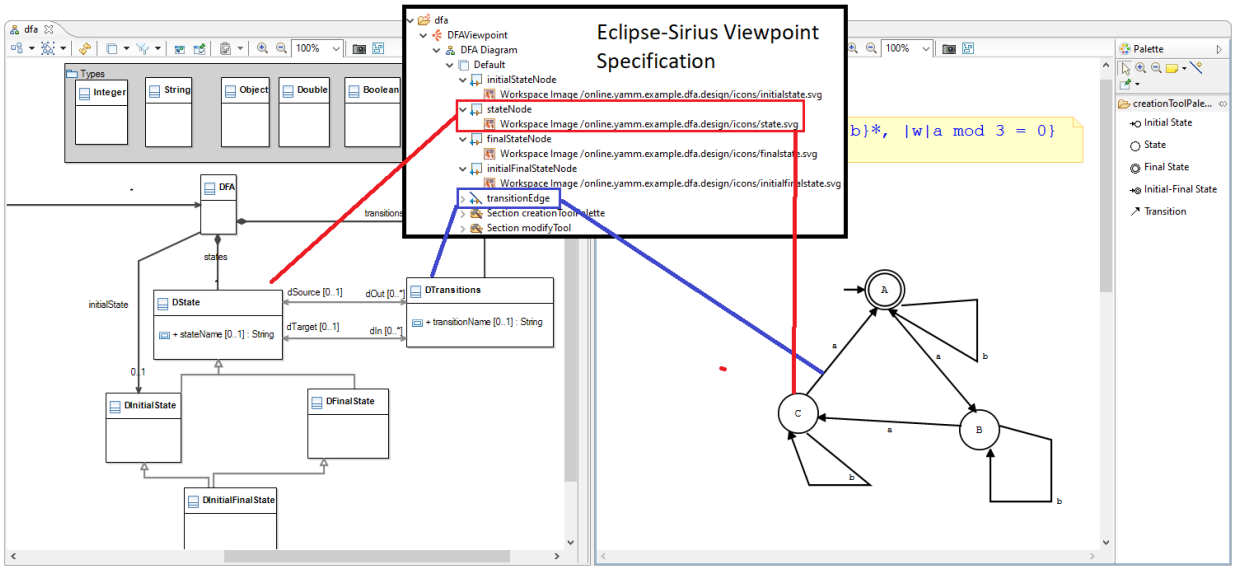

Abbildung 5: Definition eines Layer-Editors mit Eclipse-Sirius

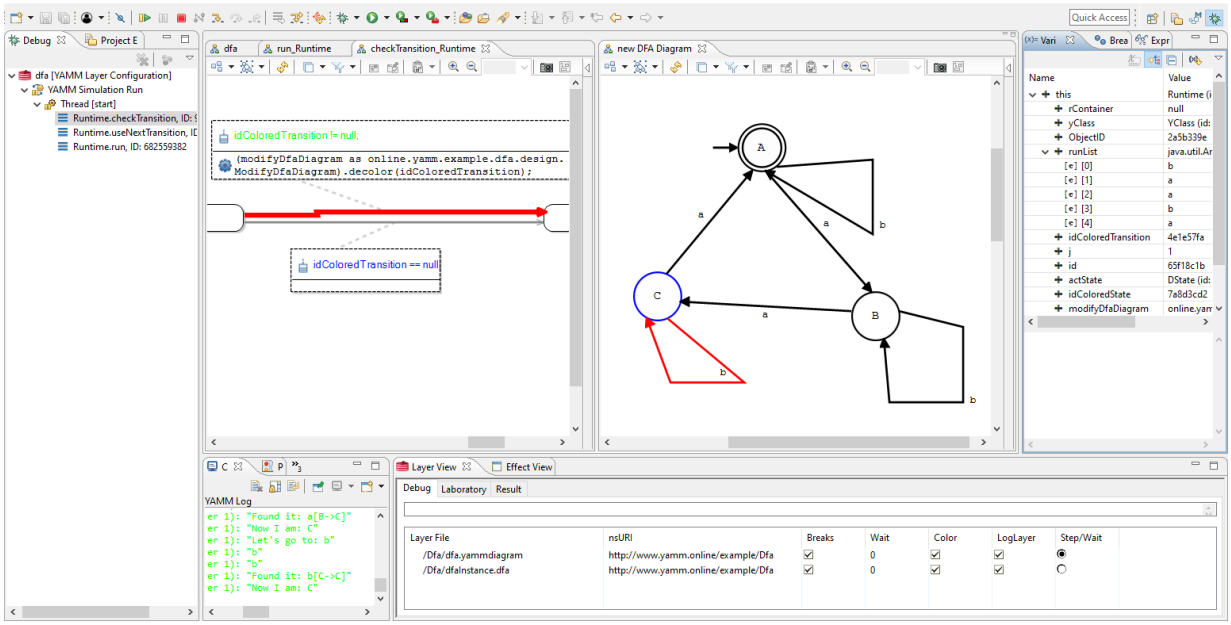

Abbildung 6: Layer-Editor mit Ausführungssemantik und Animationsansteuerung

YAMM lässt sich mit Editoren für ausführbare Modellierungssprachen erweitern (Layer-Editor). Dazu wird mit Eclipse-Sirius eine Grafische Notation für die im YAMM-Klassendiagramm Modellierten Klassen definiert und der Erzeugte Editor mit Animationsfähigkeiten erweitert. Die mit dem Layer-Editor erstellten Modelle werden beim Simulationsstart geladen und die Animationen über Effekte im YAMM-Modell angesteuert. Über die Integration in den Debugger lässt sie die Simulation auf der YAMM-Ebene, sowie auf der Layer-Editor Ebene untersuchen.

Mehrere Beispielmodelle als Vorlage zum Erstellen von Layer-Editoren sind in YAMM mitgeliefert:

- Deterministic finite automaton (DFA)
- Markov-Ketten

Im Rahmen einer Bachelorarbeit wurde ein Business Process Model and Notation(BPMN)-Layer-Editor erstellt.

6 Discrete-Event-System

YAMM enthält ein Discrete-Event-System bei dem der Programmablauf über das Senden und Empfangen von Events gesteuert wird. Die Events lösen zu ihrem Eintrittszeitpunkt eine Operation der Simulationsobjekte (Geldautomat und Kunde in Abb. 7) aus und diese ändern daraufhin den Zustand des Systems. Dadurch kann die Simulation vom Eintreten eines Events zum nächsten springen (diskrete Zeit). Simulationsobjekte können zudem andere Simulationsobjekte aus einer vorgeschalteten Warteschlange empfangen.

Um aussagekräftige bzw. realitätsnahe Modelle zu entwerfen können Werte über Zufallsverteilungen variiert werden. Dazu sind eine Reihe von vorgefertigten Zufallsverteilungen enthalten. Zur Ermittlung von gemittelten Resultaten und zum Vergleich von Parametervarianten wird die Simulation über die Laboratory-View mehrfach hintereinander und parallel ausgeführt (lokal und verteilt im Netzwerk auf mehreren Rechner). Die Gesammelten Resultate/Metriken, wie Geschwindigkeit und Auslastung der Simulationsobjekte wird in der Result-View angezeigt und kann zu weiteren Verarbeitung exportiert werden (im CSV Format zum erstellen von BIRT-Berichten und zur Verwendung in externen Programmen).

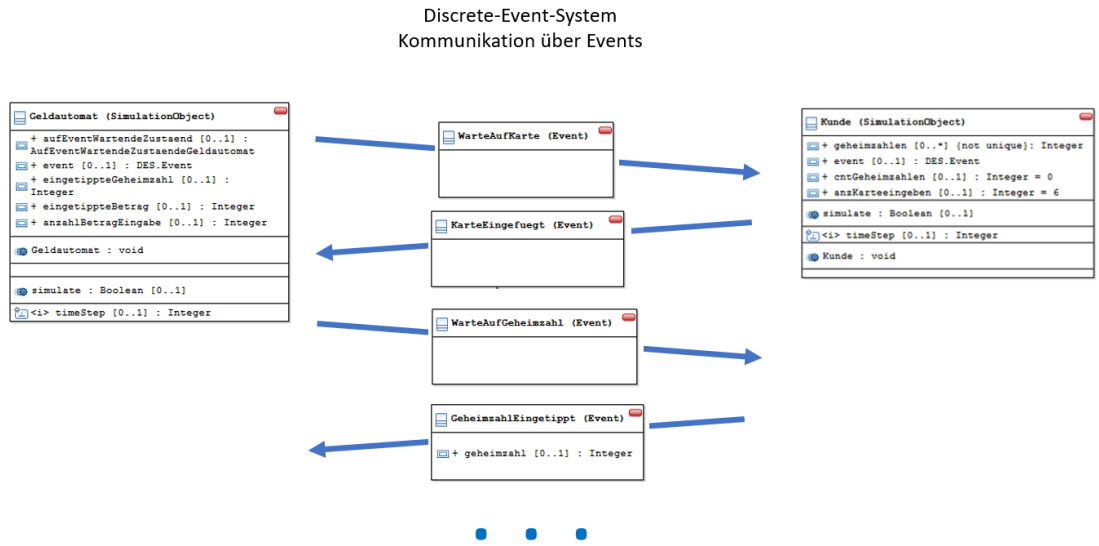

Abbildung 7: Kommunikation über Event

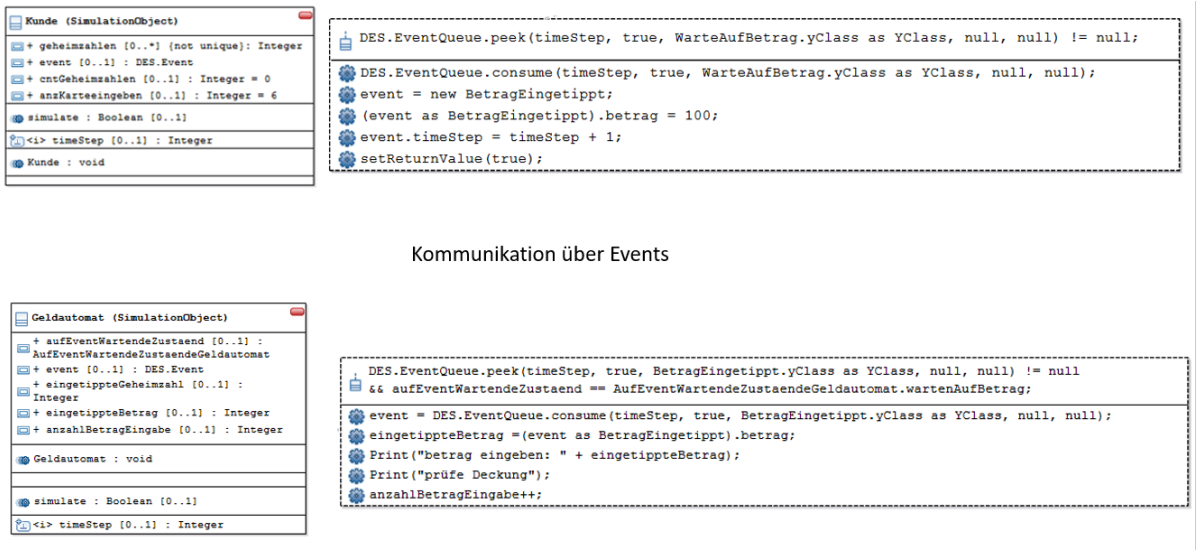

Abbildung 8: Emittieren und Konsumieren von Events