Sirius-Editor für erweiterte Datenflussdiagramme

Katrin Bott

22. September 2020

Institut für Programmstrukturen und Datenorganisation (IPD)
Praktikum: „Werkzeuge für agile Modellierung “
Betreuender Mitarbeiter: M.Sc. Stephan Seifermann

1 Einführung

Datenflussmodelle beschreiben Systeme aus einer funktionalen Sicht mittels ausgetauschter Daten. Sie finden unter anderem im Requirements Engineering als auch in der Sicherheitsanalyse ihre Anwendung. In einem klassischen Datenflussdiagrammmodell wird keine Unterscheidung getroffen *welche Eigenschaften* die transportierten Daten haben.

In dem kleinen Anwendungsbeispiel in Abbildung 1 lässt sich gut veranschaulichen, dass sich die Daten insbesondere im Bezug auf ihre Vertraulichkeit deutlich unterscheiden. Die Kreditkarteninformationen eines Buchenden sind deutlich schützenswerter als die öffentlich zugänglichen Daten eines Flugs. Mit einem erweiterten Datenflussmodell lassen sich Charakteristiken, wie hier die Zugriffsrechte, repräsentieren und erlauben somit mehr Aussagen über die Sicherheitseigenschaften geplanter Systeme zu treffen.

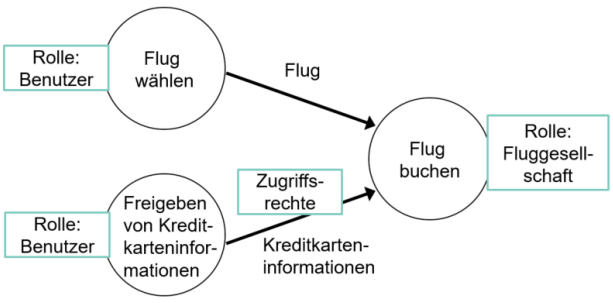

Abbildung 1: Datenflussdiagramm für eine Flugbuchung

Daraus ergibt sich die Aufgabenstellung einen Sirius-Editor für erweiterte Datenflussdiagramme zu entwickeln. Im letzten Semester wurde ein Metamodell und ein graphischer Sirius-Editor für Datenflussdiagramme entwickelt, der die Verfeinerung von Prozessen und Datenflüssen unterstützt. Dieser Editor sollte im Rahmen dieses Praktikums um verschiedene neue Elemente wie

- die Charakterisierung der Knoten
- Pins zum Datenaustausch
- Verhaltensbeschreibung von Knoten

erweitert werden und die Verfeinerung an das erweiterte Modell angepasst werden.

2 Grundlagen

2.1 Metamodelle

Im Folgenden werden die zugrundeliegenden Metamodelle und das Framework Eclipse-Sirius kurz erläutert.

2.1.1 Metamodell für klassische Datenflussdiagramme

Das ursprüngliche Metamodell für Datenflussdiagramme *DataFlowDiagram* und das Metamodell für die Datentypen *DataDictionary*, sowie der Sirius-Editor befinden sich hier [6]. Dabei werden die vier grundlegenden Elemente eines klassischen Datenflussdiagramms

- **Externer Aktor**: Einführen von Daten in das System
- **Prozess**: Transformation eingehender und ausgehender Datenflüsse
- **Speicher**: Lesen-/Schreiben von Daten
- **Datenflüsse**: Transport von Daten von einer Quelle zu einem Ziel

und eine Verfeinerung von Prozessen und Datenflüssen modelliert.

Die Verfeinerung eines Prozesses wird durch ein weiteres Datenflussdiagramm repräsentiert, das mit dem ursprünglichen Datenflussdiagramm verknüpft ist und erlaubt eine Navigation zwischen diesen Datenflussdiagrammen. Diese Verfeinerung kann zur näheren Beschreibung eines Prozesses genutzt werden. Falls ein Datenfluss mehrere Daten enthält oder einen Datentyp besitzt, der selbst Einträge besitzt (*CompositeDataType*) kann dieser Datenfluss verfeinert werden. Hierbei werden die Daten des Datenflusses aufgeteilt und es entsteht pro Datum ein neuer Datenfluss.
Das ursprüngliche Metamodell und die Verfeinerung wird genauer in diesem Praktikumsbericht [3] beschrieben.

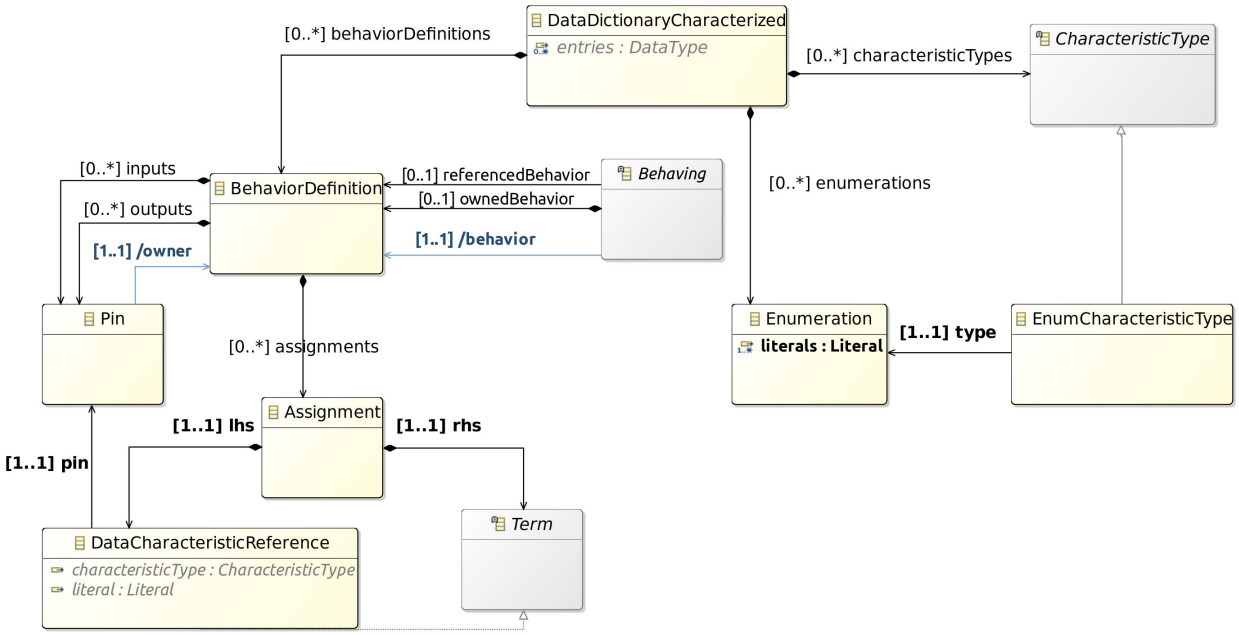

Abbildung 2: Vereinfachtes Metamodell des DataDictionaryCharacterized

2.1.2 Metamodell für erweiterte Datenflussdiagramme

Das erweiterte Modell für Datenflussdiagramme *DataFlowDiagramCharacterized* und das Metamodell für die Datentypen *DataDictionaryCharacterized*, sowie der Sirius-Editor befinden sich hier [2]. Dabei wurde insbesondere das *DataDictionary* erweitert, durch die Definition von charakteristische Typen (*CharacteristicType*) und die Definition von Verhalten (*BehaviorDefinition*), siehe Abbildung 2. Darauf aufbauend ist das Metamodell des Datenflussdiagramms (*DataFlowDiagramCharacterized* erweitert: Knoten (und Datenflüsse) können eine Menge von Charakteristiken (die jeweils einen charakteristischen Typen haben) besitzen, durch die sie näher beschrieben werden können (→ *CharacterizedProcess, CharacterizedExternalActor, CharacterizedStore, CharacterizedDataFlow*).

Ein Knoten besitzt *genau ein* Verhalten, entweder kann eine bestehende Verhaltensdefinition referenziert werden (*referencedBehavior*) oder ein eigenes Verhalten definiert werden (*ownedBehavior*). In einer Verhaltensdefinition befindet sich die Menge der In- und Output-Pins und die Definition von Zuweisungen (*Assignment*). Die Datenflüsse verlaufen von Output zu Input-Pins. Die Zuweisungen bestehen aus einer rechten und einer linken Seite. Die linke Seite bezieht sich auf einen Pin und kann einen charakteristischen Typen und ein Literal besitzen, die rechte Seite ist ein logischer Ausdruck.

2.2 Eclipse-Sirius

Eclipse-Sirius ist ein Framework zur Erstellung von graphischen Editoren und basiert sowohl auf dem Eclipse Modelling Framework (EMF). Die Grundidee basiert auf einer logischen Trennung zwischen der semantischen Information, dem Metamodell, und der

graphischen Repräsentation der Modellelemente durch den Editor.

2.2.1 Grundlegende Bestandteile von Sirius-Editoren

Hier ist ein kleiner Überblick über die grundlegendsten Bestandteile von Sirius-Editoren. Eine nähere Erläuterung der Aspekte von Sirius lassen sich in der Dokumentation nachlesen. [4]

- **Sirius-Elemente**: Mapping einer graphischen Repräsentation auf ein semantisches Element, wie z.B. Knoten oder Kanten; je nach Zustand der Elemente können verschiedene Styles festgelegt werden
- **Tools**: Festlegen von verschiedenen Aktionen bzw. Verhalten des Editors mittels Operationen oder Java Services z.B. das Erstellen von Knoten oder Kanten, Definieren von Doppelklicks oder Festlegen bei der Löschung eines graphischen Elements passiert
- **Properties-View:** Definition von eigenen Pages („Tabs“) für die Properties-View, beziehen sich auf Untermenge von Elementen und können z.B. genutzt werden um komplexere semantische Veränderungen darzustellen

Die empfohlene Sprache für Queries bzw. interpretierbare Ausdrücke für Sirius ist AQL (Acceleo Query Language), z.B. zur Bestimmung auf welches semantische Element sich ein Mapping bezieht (*Semantic Expression*). AQL kann jedoch nicht zur Modifizierung des Metamodells genutzt werden. Dafür können Operationen definiert werden, wie z.B. das Erstellen einer neuen Objektinstanz oder das Navigieren durch Kontexte. Um komplexere Änderungen am Metamodell vorzunehmen, können Java Services definiert werden.

2.2.2 Diagrammerweiterungen

Ein bestehendenes Diagramm lässt sich durch die Definition eines *Diagram Extension Points* erweitern. [5] Dabei können bestehenden Mappings aus dem ursprünglichen Diagramm wiederverwendet werden, indem sie importiert werden und für den erweiterten Editor spezialisiert bzw. modifiziert werden, z.B. durch das Definieren neuer Styles oder das Hinzufügen angrenzender Knoten (*Bordered Node*). Über die Viewpoint Selection kann die Erweiterung aktiviert bzw. deaktiviert werden.

3 Ergebnis

In diesem Praktikum wurde eine solche Diagrammerweiterung für einen Sirius-Editor umgesetzt. Die Hauptaspekte werden im Folgenden erläutert.

3.1 Graphische Repräsentation

In Abbildung 3 sieht man wie das Einführungsbeispiel im erweiterten Sirius-Editor dargestellt wird. Für die Repräsentation von charakterisierten Prozessen, Externen Aktoren

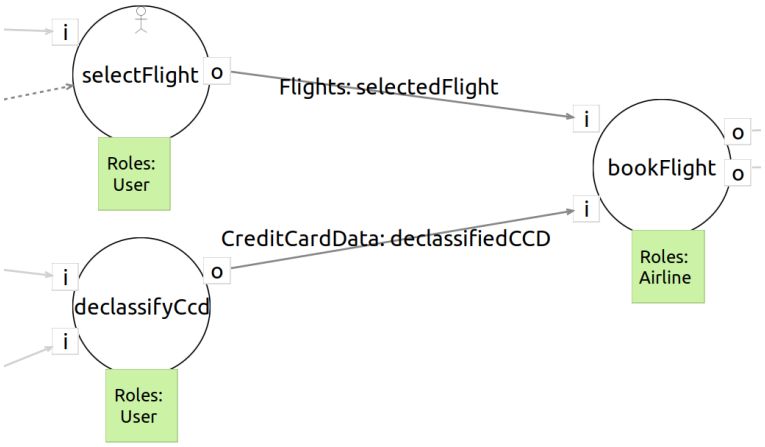

Abbildung 3: Ausschnitt aus dem Sirius-Editor

und Speichern wurden die jeweiligen Mappings aus dem ursprünglichen Editor importiert und jeweils durch angrenzende Knoten erweitert: die In- und Output-Pins und die Charakteristiken. Die charakterisierten Datenflüsse verlaufen von einem Output-Pin zu einem Input-Pin.

3.2 Erweitern der Properties-View

Die Diagrammerweiterung übernimmt die ursprünglich definierte Properties-Ansicht des DataFlowDiagram-Editors. Durch das erweiterte Metamodell gibt es neue Elemente, für die noch keine Properties-View definiert war, wie die Charakteristiken oder die Verhalten, oder erweiterte Elemente, für die die ursrprüngliche Properties-View angepasst werden musste, wie z.B. für die Datenflüsse. Das Erweitern der Properties-View lässt sich nicht wie eine Diagrammerweiterung über einen Extension Point definieren. Man kann aber die einzelnen Pages bzw. Gruppen (einzelne Abschnitte innerhalb einer Page) erweitern (*extend*) oder überschreiben (*override*). Wenn man mittels *Extend* die ursprüngliche Page erweitert, entsteht eine neue Page, bei der die ursprünglichen Gruppen übernommen und die neu definierten Gruppen hinzugefügt werden. Dabei bleibt aber die ursprüngliche Page bestehen anstatt dass nur eine gemeinsame erweiterte Page existiert (siehe Abb. 4). Dieses Problem lies sich nicht durch die Dokumentation aufklären und könnte vermutlich daran liegen, dass die ursprüngliche Page die über den Diagram Extension Point übernommen wird, bestehen bleibt.
Ein anderer Ansatz war es die Page mittels *Override* zu überschreiben, dabei entsteht zwar eine gemeinsame Page, die neu definierten Gruppen werden aber nicht angezeigt. Möglicherweise besteht ein Problem seitens Sirius, was die Erweiterung oder das Überschreiben von Pages in der Properties-View angeht, wenn Diagram Extension Points genutzt werden.

Abbildung 4: Properties-View für einen charakteristierten Prozess mit zwei *Edit* Pages

3.3 Darstellung der Verhalten

Schon bei der Betrachtung des vereinfachten Metamodell des *DataDictionaryCharacterized* (siehe Abb. 2), erkennt man, dass die Definition eines Verhaltens relativ komplex ist. Die In- und Output-Pins lassen sich zwar gut grafisch darstellen. Für die Zuweisung der Verhalten hingegen stellte sich eine graphische Darstellung als nicht sonderlich geeignet dar, insbesondere auch im Hinblick darauf die Verhalten zu bearbeiten.

Zusätzlich muss auch unterschieden werden, ob ein Knoten ein *referencedBehavior* oder ein *ownedBehavior* besitzt. Referenziert ein Knoten nur ein Verhalten sollte das Verhalten nicht von dem Knoten aus bearbeitbar sein, bei einem ownedBehavior wiederum schon. So lassen sich auch nur neue Pins zu einem Knoten mit einem ownedBehavior hinzufügen, aber nicht zu einem Knoten, der ein Verhalten referenziert. Die Repräsentation der Verhaltenszuweisungen ist aktuell eher rudimentär umgesetzt (vgl. Abb. 4):
die Elemente der linken Seite, das Pin auf das sich das Verhalten bezieht, das Literal und der Charakteristische Typ werden aufgelistet, die Terme aus der rechten Seite der Zuweisung noch nicht. So kann man hier sehen, dass die Zugriffsrechte aus dem Anfangsbeispiel, die man für die Kreditkarteninformationen benötigt, als Charakteristik innerhalb einer Verhaltenszuweisung modelliert werden. Diese Verhaltenszuweisung bezieht sich wiederum auf das Output-Pin des charakterisierten Prozesses „declassifyCcd“, von dem aus der Datenflusses beginnt für den diese Zugriffsrechte benötigt werden. (vgl. Abb. 3)

Wünschenswert wäre ein textueller Editor, der sowohl zur Anzeige als auch der Bearbeitung der Verhalten dient. Ein Ansatz ist es einen Xtext-Editor mittels der Xtext-Sirius-Integration [1] zu integrieren, dazu reichte der zeitliche Rahmen des Praktikums aber nicht mehr aus.

3.4 Anpassung der Verfeinerung an das erweiterte Metamodell

Die Herausforderung bei der Verfeinerung war es die Verhalten und Pins zu berücksichtigen. Die Verfeinerung von Prozessen erforderte wieder das Importieren von Mappings und ist unter Berücksichtigung der neuen Elemente des erweiterten Metamodells weitestgehend analog umgesetzt.

Bei der Verfeinerung der Datenflüsse haben die Verhalten eine noch größere Relevanz. Wenn ein Datenfluss verfeinert wird, dann wird der ursprüngliche Datenfluss durch mehrere neue Datenflüsse ersetzt. Da durch einen Pin nur ein Datenfluss ein- oder ausgehen

darf, müssen pro Datum jeweils ein neuer Input- und ein neuer Output-Pin erstellt werden. Dabei stellt sich die Frage, wie man mit den Verhalten, die die ursprünglichen Pins referenzieren und den Verhalten der neu erstellten Pins, umgeht. Ein reines Kopieren der Verhalten erzeugt wahrscheinlich nicht das gewünschte Verhalten, gleichzeitig geht aber bei einer reinen Ersetzung das Verhalten der ursprünglichen Verhalten verloren. Hierbei wird also eine manuelle Änderung der Verhalten erforderlich, wobei wieder ein textueller Editor von Vorteil wäre.

4 Zusammenfassung und Future Work

Im Rahmen dieses Praktikums wurde ein bestehender Sirius-Editor für Datenflussdiagramme erweitert, um die Repräsentation von der Charakterisierung von Knoten und den Verlauf der Datenflüsse durch Pins. Die Verfeinerung der Prozesse und Datenflüsse wurde soweit wie möglich an das erweiterte Modell angepasst. Hierbei wurden einige konzeptionelle Herausforderungen ausgearbeitet, die sich durch das komplexere Metamodell ergeben: Das Anpassen der Verhalten bei der Verfeinerung und das Bearbeiten von Verhalten.

Dadurch ergeben sich Ansatzpunkte für zukünftige Arbeiten:
Einer der wichtigsten nächsten Schritte wäre es den Sirius-Editors um einen textuellen Xtext-Editor zur Darstellung und Bearbeitung der Verhalten zu erweitern. Ein nächster Punkt ist das Bearbeiten der Verhaltenszuweisungen bei der Erstellung neuer Pins bei der Verfeinerung der Datenflüsse und eine Repräsentation der Verhaltens, die an Pins geknüpft sind, die bei der Verfeinerung ersetzt werden und damit nicht mehr im Modell existieren. Auch die Verfeinerung von Datenflüssen zwischen den Ebenen ist ein Ansatzpunkt, der noch berücksichtigt werden sollte.

Literatur

[1] Niko Stotz und Joost van Zwam. *altran-mde/xtext-sirius-integration*. `https://github.com/altran-mde/xtext-sirius-integration`. 2019.

[2] Stephan Seifermann und Katrin Bott. *Trust40-Project/Palladio-Supporting-DataFlowDiagramConfidenti* `https://github.com/Trust40-Project/Palladio-Supporting-DataFlowDiagramConfidentiality` 2020.

[3] Simon Schwarz. *Praktikumsbericht „Entwicklung eines Editors für Datenflussdiagramme“*. `https://github.com/Trust40-Project/Palladio-Supporting-DataFlowDiagram/blob/master/bundles/org.palladiosimulator.dataflow.diagram.editor.sirius/report.pdf`. 2020.

[4] *Sirius Dokumentation*. `https://www.eclipse.org/sirius/doc/`. 2020.

[5] *Sirius Dokumentation Diagrammerweiterung*. `https://www.eclipse.org/sirius/doc/specifier/diagrams/Diagrams.html#extensibility`. 2020.

[6] Simon Schwarz und Stephan Seifermann. *Trust40-Project/Palladio-Supporting-DataFlowDiagram*. `https://github.com/Trust40-Project/Palladio-Supporting-DataFlowDiagram`. 2020.