Visualisierung von SCADE-Modellen in KLighD

Axel Umland

Studienarbeit
eingereicht im Jahr 2014

Christian-Albrechts-Universität zu Kiel
Institut für Informatik
Arbeitsgruppe für Echtzeitsysteme und Eingebettete Systeme
Prof. Dr. Reinhard von Hanxleden

Betreut durch: Dipl.-Inf. Christian Schneider

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Kiel,

Zusammenfassung

Das Erstellen von graphischen Modellen ist eine hilfreiche Methode zum besseren Verständnis von Programmen und Funktionen. Insbesondere bei komplexen Modellen kann das manuelle Erstellen von Diagrammen sehr zeitaufwendig werden. Es existieren Werkzeuge mit denen der Zeitaufwand verringert werden kann indem Arbeitsschritte automatisiert werden.

Ziel dieser Arbeit ist es ein Eclipse-Plugin zu entwickeln, dass die Daten von manuell erstellten SCADE-Modellen in das KIELER Actor Oriented Modeling (KAOM) Format migriert. Die migrierten Daten sollen anschließend mithilfe von KIELER Lightweight Diagrams (KLighD) visualisiert werden. Dabei soll es möglich sein die Modelle im originalgetreuen Layout und einem von KLighD automatisch generierten Layout anzeigen zu lassen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

EMF Eclipse Modeling Framework

KAOM KIELER Actor Oriented Modeling

KAOT KIELER Actor Oriented Textual-Modeling

KIELER Kiel Integrated Environment for Layout Eclipse RichClient

KLighD KIELER Lightweight Diagrams

SCADE Safety-Critical Application Development Environment

XMI XML Metadata Interchange

XML Extensible Markup Language

Kapitel 1

Einleitung

Die Verwendung von graphischen Modellen zur Visualisierung von Programmen und Funktionen ist weit verbreitet [FvH10]. Solche Diagramme erleichtern erheblich das Verständnis von komplexen Modellen und Funktionen.

Das manuelle Erstellen von Diagrammen kann mitunter allein aufgrund der Größe der darzustellenden Elemente sehr zeitaufwendig werden. Es existieren diverse Werkzeuge, die das Erstellen von Diagrammen erleichtern in dem sie gewisse Arbeitsschritte übernehmen, zum Beispiel die Anordnung der Elemente oder das automatische Verbinden von Elementen mit Kanten, bis hin zu vollständig automatisch generierten Layouts.

1.1. Zielstellung

Mithilfe des von Esterel Technologies[1] entwickelten Safety-Critical Application Development Environment (SCADE) lassen sich modellbasierte eingebettete Softwaresysteme erstellen [Ber07]. Diese Modelle lassen sich mit der Entwicklungsumgebung auch simulieren und validieren. Anwendung finden die entwickelten Systeme unter anderem in der Automobil- und Eisenbahnindustrie.

Ziel dieser Arbeit ist es, mit SCADE erstellte Modelle in das KIELER Actor Oriented Modeling (KAOM) Format zu migrieren und anschließend die Diagramme mithilfe von KIELER Lightweight Diagrams (KLighD) zu visualisieren. Die Migration der Daten ist notwendig, da die Werkzeuge zur Erstellung von SCADE-Modellen nicht frei zur Verfügung und weiteren Bearbeitung stehen.

Dabei sollen neben den semantischen Daten auch alle originalen graphischen Informationen der Diagramme mit übernommen werden. Dadurch lässt sich bei der Visualisierung das originale Layout einem automatisch generierten Layout gegenüber stellen und vergleichen. Ein weiteres Ziel ist die Implementierung verschiedener Darstellungsoptionen für die Visualisierung. Somit kann die Darstellung dieser SCADE-Modelle

[1] http://www.esterel-technologies.com/

in KLighD als ein weiterer Benchmark für die automatische Layoutgenerierung und deren Darstellung dienen.

Als Testdaten für die Migration und Visualisierung der Modelle liegen dieser Arbeit die Ergebnisse des Modelleisenbahnprojekts 2007 am Institut für Informatik, Arbeitsgruppe für Echtzeitsysteme und Eingebettete Systeme der Christian-Albrechts-Universität zu Kiel zugrunde.

1.2. Aufbau dieser Arbeit

Zunächst erfolgt in Kapitel 2 eine kurze Vorstellung der SCADE Sprache und des KAOM-Formats. Anschließend wird die Migration von Modellen von SCADE nach KAOM beschrieben. Danach wird in Kapitel 3 die Visualisierung dieser Modelle mithilfe von KLighD erläutert. Abschließend wird in Kapitel 4 auf einschränkende Faktoren, sowie auf Erweiterungsmöglichkeiten eingegangen.

Kapitel 2

Migration von SCADE-Modellen in das KAOM-Format

Der erste Teil dieser Studienarbeit behandelt die Migration von SCADE-Modellen in das KAOM-Format. Es folgt eine kurze Vorstellung der beiden Formate, sowie eine Beschreibung wie die SCADE-Modelle im KAOM-Format repräsentiert werden sollen. Anschließend folgt eine Beschreibung der Umsetzung dieser Datenmigration.

Die Migration der Modelle ist notwendig, da die Daten im SCADE-Format nicht frei veröffentlicht werden können. Das Erstellen der SCADE-Modelle erfolgte mittels einer akademischen Lizenz. Ferner stellen die Entwickler der SCADE-Werkzeuge eine Möglichkeit des Zugriffs auf die Daten mittels des Eclipse Modeling Framework (EMF) bereit. Es lassen sich somit Werkzeuge der Eclipse Entwicklungsumgebung benutzen und Plugins erstellen, mit denen die Modelle geladen und weiterverarbeitet werden können.

2.1. Das SCADE-Format

Mit der SCADE-Entwicklungsumgebung lassen sich beliebig tief verschachtelte Datenfluss- und Zustandsdiagramme erstellen. Diese SCADE-Modelle sind im XMI Format gespeichert. Das Erstellen dieser Modelle kann sowohl mittels einer graphischen als auch einer textuelle Entwicklungsumgebung erfolgen. In Abbildung 2.1 ist ein SCADE-Operator mit drei Eingängen zu sehen, welcher mit einer Vergleichsoperation und einer Fallunterscheidung ein Ausgangsdatum berechnet.

Im SCADE-Format werden zum einen die semantischen Informationen des Modells, im folgenden abstrakte Daten genannt, und zum anderen die konkreten Daten aller im Modell enthaltenen Elemente gespeichert. Die abstrakten Daten beinhalten die einzelnen Gleichungen (`equations`) aller Elemente, die Ein- und Ausgänge, alle lokalen Variablen, sowie Ein- und Ausgangssignale. Die konkreten Datensätze beinhalten alle in der graphischen Darstellung vorhandenen Elemente. Diese graphischen Darstellungen beschreiben Blockdiagramme, welche in der SCADE-Syntax als Netzdiagramme bezeich-

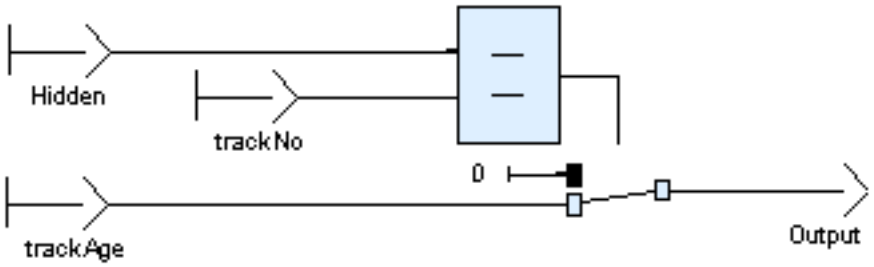


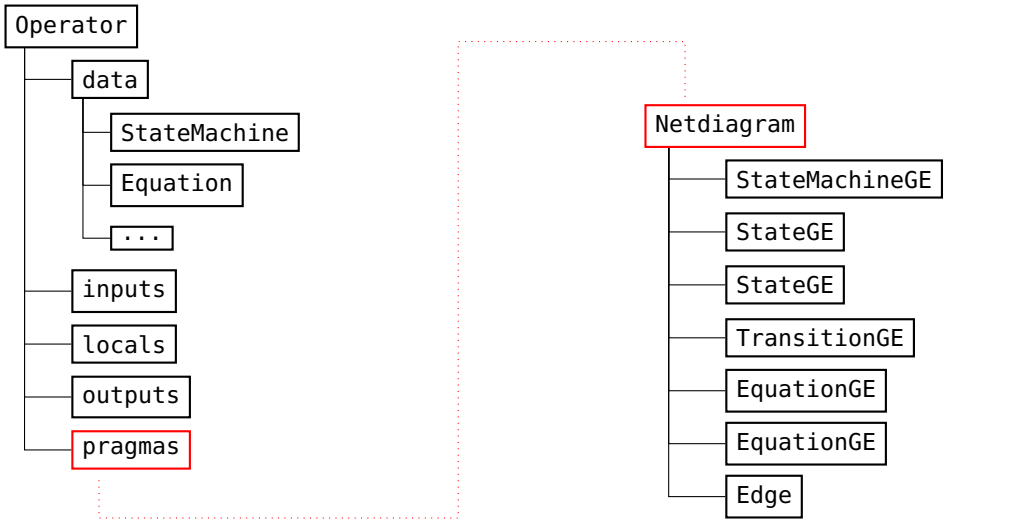
Abbildung 2.1. Beispieloperator "AgeRemoveTrack" in SCADE

net werden. Alle Elemente dieser Diagramme werden mit den konkreten Positions-, Größen- und Rotationsangaben in den konkreten Datensätzen des Modells gespeichert. Für jedes Element werden zusätzlich alle ein- und ausgehenden Kanten aufgelistet. Ebenfalls enthalten sind alle Kanten einschließlich der konkreten Positionsangaben aller Knickpunkte dieser Kanten in der graphischen Darstellung. Aufgrund einer besseren Lesbarkeit kann es vorkommen, dass einzelne Elemente doppelt im Diagramm vorkommen (beispielsweise Eingangssignale). Dies hat zum Beispiel den Vorteil, dass es weniger Kantenüberschneidungen gibt. Diese doppelten Elemente werden in den konkreten Datensätzen mit abgelegt, tauchen in den abstrakten Daten jedoch nicht doppelt auf. Zusätzliche Informationen, wie zum Beispiel Kommentare, können für jedes Modell und die jeweiligen einzelnen Elemente vorhanden sein.

Der Aufbau der Datenstruktur ist weitestgehend flach. Bei den konkreten Daten werden alle Gleichungen in einer Liste gespeichert. Das bedeutet, dass alle Elemente innerhalb eines Netzdiagramms, egal welchen Typs und welcher Verschachtelungstiefe, das gleiche übergeordnete Element (in diesem Fall das Netzdiagramm) haben. In Abbildung 2.2 ist zur Veranschaulichung ein Beispiel der Datenstruktur eines Zustandsdiagramms mit je zwei Zuständen und Transitionen dargestellt. Dabei sind in Abbildung 2.2a der Aufbau der abstrakten Daten veranschaulicht und in Abbildung 2.2b das `Netdiagram` mit den konkreten Datensätzen.

2.2. Das KAOM-Format

Das KAOM-Format ist ein freies Format und dient zur Beschreibung von Modellen. Die KAOM-Modelle werden ebenfalls im XMI Format gespeichert. Die Syntax des KAOM-Formats ist sehr einfach aufgebaut. Es gibt als Klassentypen `entities`, `ports`, `links` und `relations`. Darüber hinaus können beliebig viele Zusatzinformationen als Annotationen an diesen Elementen gespeichert werden. Das KAOM-Meta Modell ist in Abbildung 2.3



(a) SCADE-Operator **(b)** Detailansicht des Netzdiagramms

Abbildung 2.2. Datenstruktur eines SCADE-Modells

veranschaulicht. Der Aufbau der KAOM-Datenstruktur ist hierarchisch. Das bedeutet,

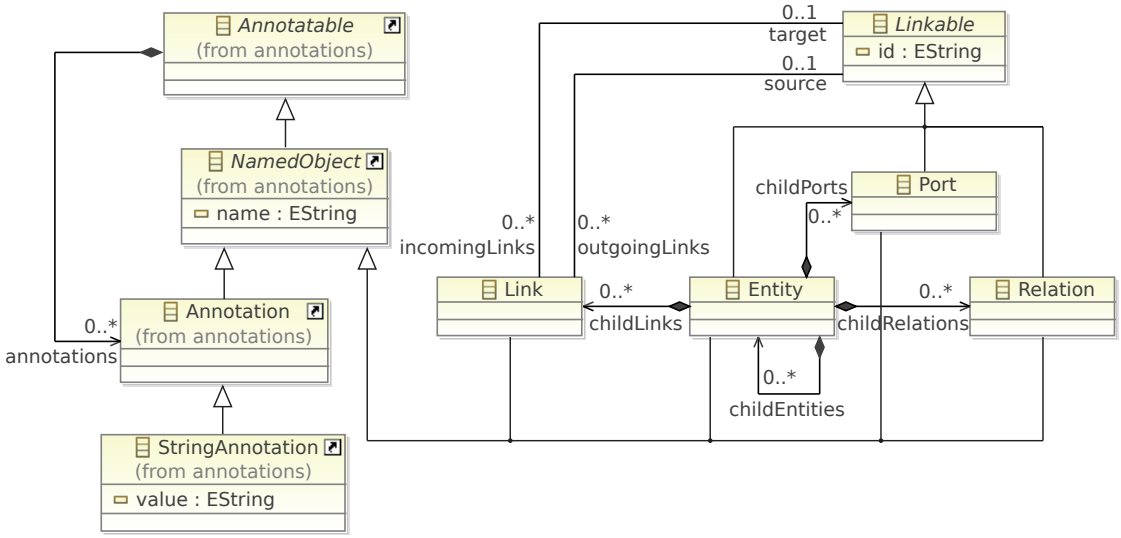


Abbildung 2.3. Das KAOM-Meta Modell, Quelle: [FvHK+14, Seite 10]

dass eine `entity` wiederum beliebig viele `entities` als Kindelemente haben kann. Diese Eigenschaft entsteht durch die reflexive Aggregation `childEntities` der `entity` Klasse. Bei verschachtelten Diagrammen lassen sich somit wesentlich schneller alle Elemente eindeutig zuordnen, zum Beispiel welcher Zustand zu welchem Zustandsdiagramm gehört. Zusätzlich zur XMI basierten Serialisierung gibt es die Möglichkeit eine deutlich

lesbarere Variante zu verwenden, das KAOT Format. Die Editierung, beziehungsweise das Erstellen von neuen Modellen in dieser textuellen Variante ist gegenüber der XMI Struktur einfacher.

2.3. Repräsentation der SCADE-Modelle in KAOM

Die Auswertung der SCADE-Modelle erfolgt durch Traversierung der konkreten Daten. Diese Daten sind zunächst notwendig um das originale Layout später in KLighD wiederzugeben. Desweiteren referenziert in der SCADE-Datenstruktur jedes Element der graphischen Repräsentation auf den jeweiligen abstrakten Datensatz, so dass die Auswertung aller benötigten Daten gegeben ist.

Die hier beschriebene Übersetzung erzeugt für jedes Netzdiagramm eines SCADE-Modells ein neues Modell im KAOM-Format. Diese Netzdiagramme enthalten eine Liste von `presentation elements`, welche Instanzen einer der fünf Klassen `EquationGE`, `Edge`, `StateMachineGE`, `StateGE` oder `TransitionGE` sind, siehe dazu Abbildung 2.2b. In Tabelle 2.1 wird beschrieben welche SCADE-Elemente welcher KAOM-Klasse bei der Datenmigration zugeordnet werden.

Tabelle 2.1. Repräsentation der SCADE-Elementtypen im KAOM-Format

SCADE-Elementtyp	KAOM-Klasse
Zustandsmachinen, Zustände und Gleichungen (StateMachineGE, StateGE, EquationGE)	KAOM entities
Kanten und Transitionen (Edge, TransitionGE)	KAOM links
Start- und Zielpunkte der Kanten	KAOM ports
Zusatzinformationen	KAOMannotations

Daraus ergibt sich die Zieldatenstuktur, wie in Abbildung 2.4 veranschaulicht.

2.4. Auswertung der SCADE-Netzdiagramme

Ein SCADE-Operator hat ein oder mehrere Netzdiagramme. Beim Import der Operatoren wird für jedes Netzdiagramm des Operators ein separates Modell im KAOM-Format generiert. Jedes Netzdiagramm enthält eine Liste von `presentation elements` welche alle Elemente des Diagramms beinhaltet. Diese Elemente sind jeweils Instanzen einer der folgenden fünf Klassentypen: `EquationGE`, `Edge`, `StateMachineGE`, `StateGE` oder `TransitionGE`.

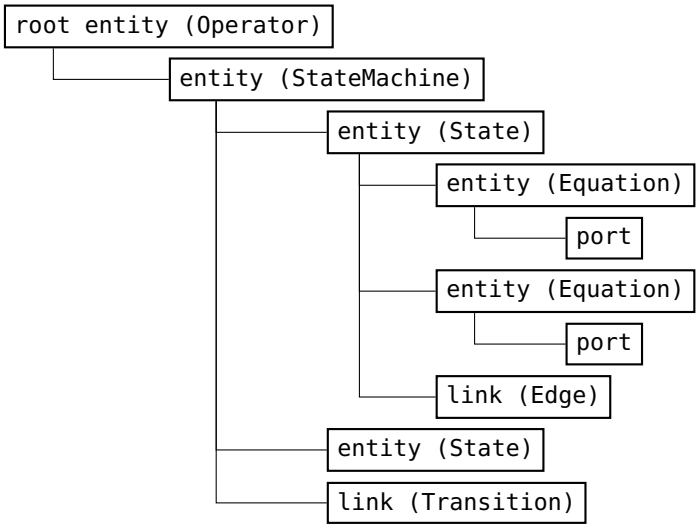


Abbildung 2.4. Zieldatenstruktur im KAOM-Format

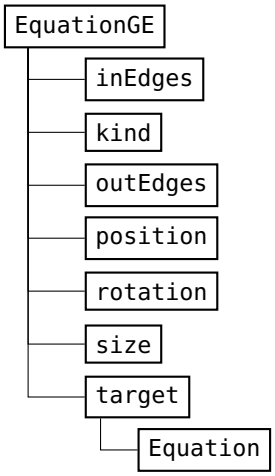
Die `root entity` im KAOM-Format entspricht dabei jedes Mal dem Operatorelement. Zusätzliche Informationen wie Typ und Kommentare werden als Annotationen an die `root entity` angehangen. Die Liste der `presentation elements` wird abgearbeitet und je nach Elementtyp werden die relevanten Informationen ausgewertet und im KAOM-Modell gespeichert.

Equations Jede `equation` steht für eine Gleichung innerhalb des Diagramms. Konkret liegt in der Liste der `presentation elements` ein `EquationGE` (equation graphical element) vor. Dieses beinhaltet zunächst alle graphischen Informationen, wie Position, Größe und Rotation. Weiterhin enthält das `EquationGE` Element zwei Listen für alle ein- und ausgehenden Kanten Das Element referenziert auf das entsprechende Gegenstück in der abstrakten Syntax, so dass alle benötigten Informationen zur späteren Visualisierung vorhanden sind und ausgewertet werden können.

Für jede `equation` des Operators wird eine `child entity` im KAOM-Format erzeugt. Alle relevanten graphischen Informationen werden als Annotationen an dieser `child entity` gespeichert. Anhand der Listen der ein- und ausgehenden Kanten der Gleichungen werden der KAOM `entity` entsprechende `child ports` hinzugefügt.

In Abbildung 2.5a ist die Struktur eines solchen Elements im SCADE-Format schematisch dargestellt, sowie deren Repräsentation im KAOM-Format (Abbildung 2.5b).

Zusätzlich wird die entsprechende `expression` der `equation` ausgewertet und deren semantische Informationen ebenfalls als Annotationen gespeichert (siehe Kapitel 2.5).



(a) Datenstruktur eines `EquationGE` Elements im SCADE-Format

```
@kind UNSPECIFIED
@inEdges_empty true
@outEdges_empty false
@posX 4075
@posY 5503
@rotation 0
@height 503.0
@width 1005.0
entity _L2 "trackNo" {
    @Port outEdge
    port out_port_L2X_L2
         "out_port_L2";
}
```

(b) Repräsentation als KAOM `entity` (KAOT Darstellung)

Abbildung 2.5. Aufbau eines `EquationGE` Elements

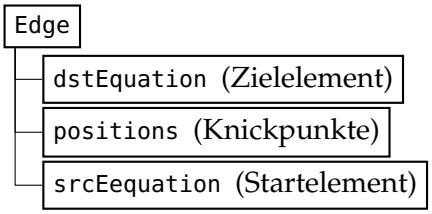
Edges Jedes Element vom Typ `Edge` in der Liste steht für eine Kante zwischen zwei `equations`. Es wird sowohl auf das Quell- und das Zielelement referenziert. Jede Kante hat ferner eine Liste von Positionsangaben, die neben dem Start- und Zielpunkt alle Knickpunkte der Kante beinhaltet. Mit diesen Knickpunkten lässt sich somit der genaue Verlauf einer Kante auslesen und zur späteren Visualisierung im Originallayout wiederverwenden.

Die Kante wird zunächst als `link` der `root entity` übergeben. Nachdem alle `presentation elements` abgearbeitet wurden, werden in einem abschließenden Schritt alle Kanten der `root entity` den entsprechenden `child entities` zugeordnet.

In Abbildung 2.6a sieht man den schematischen Aufbau eines `Edge` Elements und dessen Repräsentation als KAOM `link` (Abbildung 2.6b).

StateMachines Der Typ `StateMachineGE` steht für einen im Modell dargestellten Zustandsgraphen. Neben den bereits bekannten graphischen Informationen beinhaltet dieser eine Liste aller zugehörigen Zustände.

Der Zustandsgraph wird zunächst als `child entity` der entsprechenden `parent entity` angelegt. Die Positions- und Größenangaben werden als Annotation gespeichert. Gemäß der Liste der zugehörigen Zustände, werden entsprechende `child entities` der `entity` des Zustandsgraphen hinzugefügt. Abbildung 2.7a stellt den schematischen Aufbau eines Zustandsgraphen in SCADE dar, Abbildung 2.7b die Repräsentation im KAOM-Format.

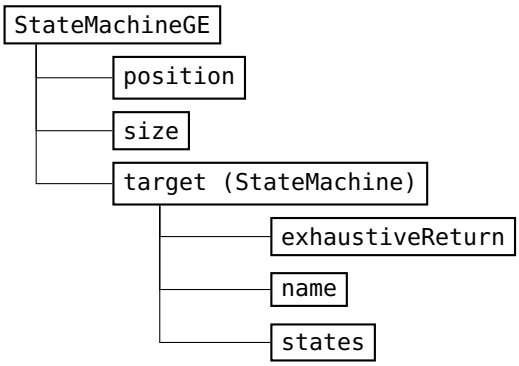


(a) Datenstruktur eines `Edge` Elements im SCADE-Format

```
@bendPoint position (
    @posX 5080.0
    @posY 5768.0
)
@bendPoint position (
    @posX 5847.0
    @posY 5768.0
)
@bendPoint position (
    @posX 6668.0
    @posY 5768.0
)
link out_port_L2X_L2
    to in_port_L2X_L3;
```

(b) Repräsentation als KAOM `link` (KAOT Darstellung)

Abbildung 2.6. Aufbau eines `Edge` Elements



(a) Datenstruktur eines `StateMachineGE` Elements im SCADE-Format

```
@localVarName SM1
@equation StateMachine
@posX 1323
@posY 9155
@rotation 0
@height 11324.0
@width 18812.0
@exhaustiveReturn false
entity SM1 "SM1" {

}
```

(b) Repräsentation als KAOM `entity` (KAOT Darstellung)

Abbildung 2.7. Aufbau eines `StateMachineGE` Elements

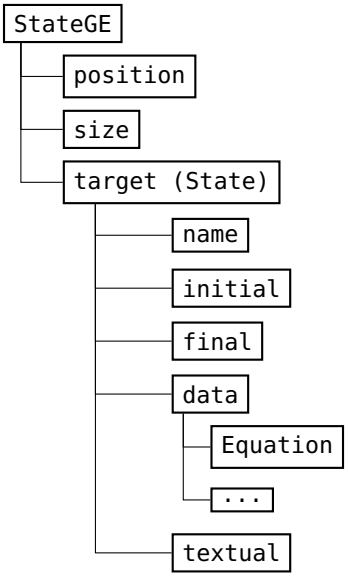
States Bei einem Element des Typs `StateGE` handelt es sich um einen konkreten Zustand innerhalb eines Zustandsgraphen. Neben den graphischen Informationen beinhaltet jeder Zustand eine Liste, welche auf alle zugehörigen `equations` der abstrakten Daten referenziert.

Die Daten der graphischen Information werden ausgelesen und als Annotationen gespeichert. Entsprechend der Liste der zugehörigen `equations` werden entsprechende `child entities` der `entity` des Zustandes hinzugefügt.

Eine Besonderheit bei Zuständen ist, dass sie auf ein separates Diagramm verweisen

können. Das kann bei verschatelten Zustandsmachinen auftreten.Ist dies der Fall, so wird ein weiteres KAOM-Modell für diese verschachtelten Diagramme in Zuständen angelegt und ausgewertet.

Der Aufbau eines `StateGE` Elements ist in Abbildung 2.8a dargestellt, sowie dessen Repräsentation im KAOM-Format in Abbildung 2.8b.



(a) Datenstruktur eines `StateGE` Elements im SCADE-Format

```
@localVarName idle_all_off
@equation State
@posX 10081
@posY 15081
@rotation 0
@height 3334.0
@width 4365.0
@isInitial false
@isFinal false
@textual false
entity idle_all_off
       "idle_all_off" {

}
```

(b) Repräsentation als KAOM `entity` (KAOT Darstellung

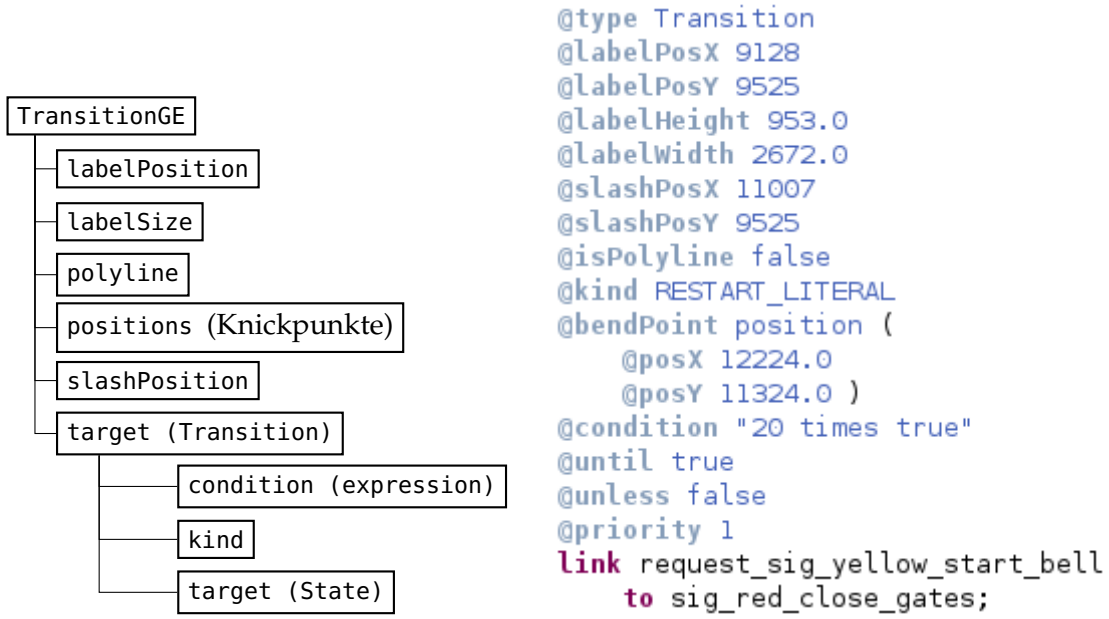
Abbildung 2.8. Aufbau eines `StateGE` Elements

Transitions Der Elementtyp `TransitionGE` spezifiziert eine Transition in einem Zustandsgraphen. Es sind sowohl die graphischen Daten der Transition und des zugehörigen Labels (Anzeige von Bedingung und Ausgabesignal) vorhanden. Die Referenzierung auf die Transition enthält neben dem Start- und Zielzustand, eine Bedingung zum Wechsel des Zustandes und optional eine Liste von Signalen die bei einem Zustandswechsel emittiert werden sollen.

Die graphischen Informationen werden wie bereits beschrieben als Annotationen an der Transition gespeichert. Für die Transition wird ebenfalls ein `link` im KAOM-Format angelget. Zu beachten ist, dass im Gegensatz zu den Kanten keine `ports` an den Start- und Zielzuständen angelegt werden. Die Werte für Start und Ziel des `links` werden direkt auf die `entities` des Start- und Zielzustandes gesetzt. Die Liste der Knickpunkte wird ausgewertet und als Annotation gespeichert.

Die Bedingung, wann von einem Zustand in einen anderen gewechselt werden

soll ist eine `expression` im SCADE-Format. Diese wird nach dem selben Muster wie bei den `equations` ausgewertet und abschließend als ein einzelner textueller Ausdruck als Annotation zu der Transition gespeichert. Der schematische Aufbau einer Transition im SCADE-Format ist in Abbildung 2.9a dargestellt. Deren Repräsentation im KAOM-Format in Abbildung 2.9b.



(a) Datenstruktur eines `TransitionGE` Elements im SCADE-Format

(b) Repräsentation als KAOM `link` (KAOT Darstellung)

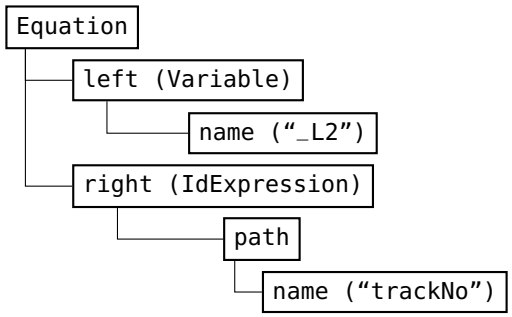
Abbildung 2.9. Aufbau eines `TransitionGE` Elements

2.5. Auswertung der SCADE-Expressions

Ein wesentlicher Bestandteil bei der Datenauswertung der `equations` ist die Evaluierung der jeweiligen `expression`. Da in den konkreten Daten in den Netzdiagrammen auf die abstrakten Datensätze verwiesen wird, kann man diese gleich mit auswerten ohne beide Listen komplett durchlaufen zu müssen. Dies hat zusätzlich den Vorteil, dass man bei der Auswertung immer gleich mit der richtigen `entity` in der KAOM-Datenstruktur arbeitet.

Bei der Auswertung wird eine Annotation mit dem Typ der `expression` angelegt und die entsprechenden Informationen der jeweiligen Elemente werden abgerufen und im KAOM-Modell abgelegt. Der lokale Variablenname wird ebenfalls als Annotation

gespeichert, genau wie der Typ und der konkrete Wert. In Abbildung 2.10a wird der schematische Aufbau einer `IdExpression` als Erweiterung des obigen Beispiels (Abbildung 2.5a) dargestellt. Eine `IdExpression` ist ein über einen Namen (`id`) referenziertes Element. In diesem Beispiel ist es das Eingangsdatum `trackNo`. In Abbildung 2.10b ist dann die vollständige Repräsentation dieses Elements im KAOM-Format zu sehen.



(a) Datenstruktur einer `Equation` im SCADE-Format

```
@kind UNSPECIFIED
@inEdges_empty true
@outEdges_empty false
@localVarName _L2
@equation IdExpression
@path trackNo
@type input
@posX 4075
@posY 5503
@rotation 0
@height 503.0
@width 1005.0
entity _L2 "trackNo" {
    @Port outEdge
    port out_port_L2X_L2
         "out_port_L2";
}
```

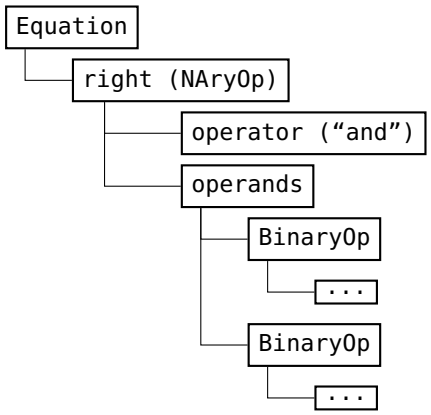
(b) Repräsentation als KAOM `entity` inklusive der Annotationen der graphischen Daten (KAOT Darstellung)

Abbildung 2.10. Aufbau einer SCADE-`expression`

Textuelle Repräsentation von Elementen Zusätzlich zur Verwendung von Piktogrammen für alle im Diagramm enthaltenen Elemente, lässt sich für (beliebig komplexe) Ausdrücke auch eine textuelle Darstellung wählen. Diese textuelle Repräsentation von Ausdrücken kann dazu beitragen, die Lesbarkeit der Diagramme zu erhöhen, insbesondere bei der Verwendung von verschachtelten Ausdrücken. Dies wird in der Annotation `kind` und dem Wert `obj_lit` (object literal) vermerkt.

Verschachtelte Ausdrücke Bei der Verwendung von textuellen Ausdrücken ist es möglich beliebig tief verschachtelte `expressions` anzugeben. Dies ist ein gutes Hilfsmittel um auch komplexe Ausdrücke platzsparend und verständlich anzuzeigen. So kann beispielsweise ein `NAryOp` (Operator mit beliebig vielen Eingängen) konkrete Eingangswerte

(Variablen) oder auch weitere Operationen als Eingangsdaten haben. In Abbildung 2.11a ist der schematische Aufbau einer `and` Operation (der `NAry` Operator) von zwei Binäroperationen (`BinaryOP`) in den semantischen Daten eines SCADE-Modells dargestellt. Dieser textuelle Ausdruck wird in den konkreten Daten als einzelnes Element gespeichert, so dass bei der Datenmigration ebenfalls eine einzelne `entity` angelegt wird (für den `NAryOp`). Die verschachtelten Binäroperationen werden wie oben beschrieben ausgewertet, nur dass keine separaten `entities` im KAOM-Modell angelegt werden. Stattdessen werden die textuellen Repräsentationen der Binäroperationen als Annotationen gespeichert. In Abbildung 2.11b sieht man einen Ausschnitt der Annotationen der `NAryOp entity`. Als `operand` Annotationen enthält diese die textuellen Ausdrücke der Binäroperationen.



(a) Datenstruktur verschachtelter Ausdrücke im SCADE-Format

```
@equation NAryOp
@operator and
@operand "out.trainNo <> 0"
@operand "out.mode = confirm"
```

(b) Repräsentation verschachtelter Ausdrücke im KAOM-Format als Annotationen (KAOT Darstellung)

Abbildung 2.11. Aufbau von verschachtelten Ausdrücken

2.6. Hinweise zur Implementierung

In diesem Abschnitt erfolgt eine Beschreibung einiger Besonderheiten der Implementierung.

Bei der Auswertung des graphischen Elements eines Zustandsgraphen wird die vorhande Liste der zugehörigen Zustände ausschließlich dazu genutzt, die entsprechenden `child entities` der korrekten `parent entity` zuzuordnen. Die weitere Auswertung aller Informationen der Zustände erfolgt erst bei Erreichen des entsprechenden `StateGE` Elements in der Liste der `presentation elements`, um sowohl die konkreten und die abstrakten Daten auswerten zu können.

Auch das `StateGE` Element beinhaltet eine Liste aller zugehörigen `equations`. Diese Liste referenziert ebenfalls auf die abstrakten Daten, so dass nur die `child entities` angelget werden. Die Auswertung dieser `equations` erfolgt erst wenn der entsprechende `equationGE` Eintrag in der Liste der `presentation elements` erreicht ist.

Beim Aufruf der Evaluierungsmethode wird dieser neben der `expression` auch ein String mit übergeben. Xtend [1] ermöglicht es über diesen String eine Fallunterscheidung durchzuführen. Der erste Aufruf der Auswertung erfolgt immer mit dem String "equation", welcher kennzeichnen soll, dass dies noch kein rekursiver Aufruf ist. Handelt es sich um einen rekursiven Aufruf wird dieser String beispielsweise durch "operand" (im Falle von verschachtelten Binäroperationen) ersetzt. Damit wird vermieden, dass Annotationen doppelt vergeben werden (zum Beispiel der Typ des Ausdrucks). Die gesamte Gleichung des verschachtelten Ausdrucks wird als String zusammengesetzt und zurückgegeben und abschließend als eine einzelne Annotation des aufrufenden Ausdrucks gespeichert, siehe Abbildung 2.11b.

Ziel dieser Variante der rekursiven Auswertung verschachtelter textueller Ausdrücke ist die Generierung einer einzelnen Annotation. Dies erleichtert die Implementierung der späteren Visualisierung, da der Text bereits in der anzuzeigenden Form gespeichert ist. Alternativ wäre auch denkbar die Annotationen im KAOM-Format zu verschachteln. Die Verschachtelung der Annotationen würde dann der Verschachtelung der Elemente im SCADE-Format entsprechen. Der Aufwand zur Implementierung wäre damit an dieser Stelle geringer, jedoch müsste man sich zur Visualisierung der Modelle weitere Methoden überlegen. Der hier bereits erstellte textuelle Ausdruck müsste dann während der Diagrammsynthese aus den verschachtelten Annotationen extrahiert und zusammengesetzt werden.

[1] `http://www.eclipse.org/xtend/`

Kapitel 3

Visualisierung mit KIELER Lightweight Diagrams

Im folgenden Abschnitt wird die Darstellung der SCADE Operatoren mithilfe des KIELER Lightweight Diagrams Framework erläutert. Die Eingabedaten sind die im KAOM-Format gespeicherten Modelle.

Nach einer kurzen Einleitung zu `KGraph`, `KRendering` und KLighD wird zunächst auf die Synthese der einzelnen Elemente eingegangen. Anschließend folgt eine Erläuterung zur detaillierteren Darstellung dieser Elemente und zum verwendeten Layout. Den Abschluss bildet eine Erläuterung der Darstellungsoptionen nach der Diagrammsynthese.

Das KGraph und KRendering Meta Modell Das `KGraph` Meta Modell repräsentiert die grundlegenden Graphenelemente Knoten, Kanten, Ports und Labels. Entsprechend gibt es die Klassen `KNode`, `KEdge`, `KPort` und `KLabel`. Das `KGraph` Meta Modell ist in Abbildung 3.1 dargestellt.

Anhand der Diagrammelemente wird ein Layoutgraph erstellt, der `KGraph` (siehe [SSvH12]). Der automatische Algorithmus zum Anordnen dieser `KGraph` Elemente ist `KLay Layered`, wie in [SSvH14] beschrieben.

`KRendering` ist eine Erweiterung von `KGraph`. Die Verwendung von `KRendering` dient dazu die Darstellung von Elementen weiter individuell anzupassen, beispielsweise durch Linienstärke oder Hintergrundfarbe. Weitere Erläuterungen dazu sind ebenfalls in [SSvH12] beschrieben.

KIELER Lightweight Diagrams Die Visualisierung der Modelle erfolgt mithilfe von KLighD. Dieses Franmework ermöglicht die Darstellung von graphenbasierten Modellen, wie in [SSvH13] beschrieben. Durch den automatischen Layoutalgorithmus wird das Erstellen der Visualisierung deutlich vereinfacht, da die einzelnen Elemente nicht manuell arrangiert werden müssen. Siehe dazu auch [FvHK+14].

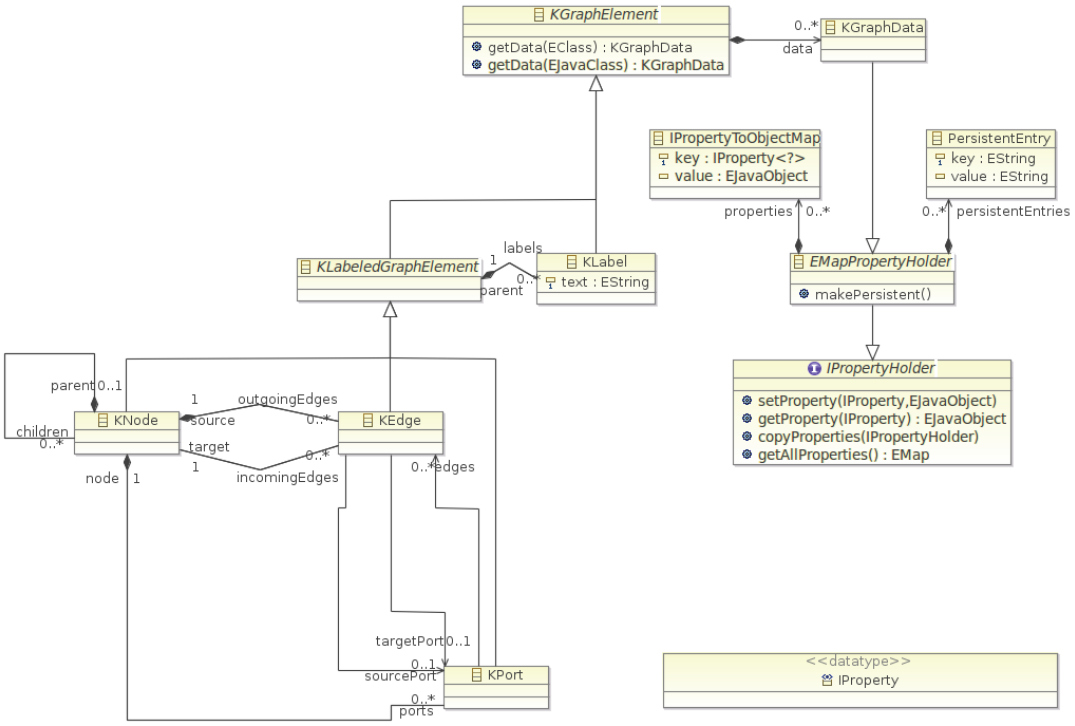


Abbildung 3.1. Das KGraph Meta Modell[1]

3.1. Die Diagrammsynthese

Die `root entity`, also der Operator, wird nicht explizit als ein graphisches Element dargestellt, stattdessen repräsentiert die gesamte Zeichenfläche den Operator. Für alle Elemente im Modell wird eine `entity` erstellt, durch Annotationen wird die graphische Darstellung bestimmt.

Entities Für jede `entity` des KAOM-Modells wird ein Knoten mit einem `RoundedRectangle` erzeugt. Dies gilt sowohl für die einzelnen `equations`, als auch für Zustandsgraphen und Zustände, da diese ebenfalls als KAOM `entites` gespeichert sind. Die minimale Größe des Knotens entspricht dabei der Originalgröße aus dem SCADE-Modell. Besitzt eine `entity` weitere `child entities` wird eine `child Area` erzeugt, die verschachtelten `child entities` werden nach dem gleichen Prinzip ausgewertet und deren Knoten innerhalb des Knotens der `parent entity` platziert. Dies gilt für die Zustände innerhalb einer Zustandsmaschine, sowie für die Gleichungen innerhalb der

[1] Quelle: `http://rtsys.informatik.uni-kiel.de/confluence/display/KIELER/KGraph+Meta+Model`

Zustände.

Die detaillierteren Informationen zur Darstellung werden durch die Auswertung der Annotationen synthetisiert und in Kapitel 3.2 erläutert.

Ports Für jeden KAOM `port` wird ein `port` im `KGraph`-Format angelegt. Im Allgemeinen werden alle `ports` für eingehende Kanten auf der westlichen Seite des Knotens und alle `ports` für ausgehende Kanten auf der östlichen Seite des Knotens festgesetzt. Ausnahmen bilden dabei Kontrollelemente wie beispielsweise `switch` und `if` Ausdrücke. Der `port` des Steuerungseingangs wird dazu auf der nördlichen Seite des Knotens festgelegt. Der Eingangsport von einem `Emission` Element wird ebenfalls auf die Nordseite des Knotens gesetzt.

Kanten Es werden alle Kanten der `entity` ausgewertet und über die generierten `ports` werden deren Start- und Zielpunkte festgelegt. Ein Knoten kann jeweils mehrere Eingangs- und Ausgangskanten mit dem selben Datum haben. Bei den Ausgangskanten wird dabei zur Förderung der Lesbarkeit auf die Verwendung mehrfacher `ports` für das gleiche Datum verzichtet. Um zu kennzeichnen, dass der Wert eines Teilausdrucks mehrfach verwendet wird, und die entsprechende Kante nicht ziellos im Raum startet, wird die Eigenschaft *junction bend points* bei den Kanten verwendet.

Transitionen werden im KAOM-Format genau wie Kanten als `links` gespeichert. Handelt es sich bei einem `link` um eine Transition zwischen zwei Zuständen, werden weitere Annotationen ausgewertet, die unter anderem die Bedingung zum Wechsel des Zustandes beinhaltet. Diese Bedingung wird als Label an die Kante mit angehangen. Wird bei einem Zustandswechsel ein Signal emittiert, so wird dies in dem Label ebenfalls angezeigt.

Abbildung 3.2 zeigt ein mit KLighD erzeugtes Diagramm des SCADE-Operators “AgeRemoveTrack” unter Verwendung des originalen Layouts.

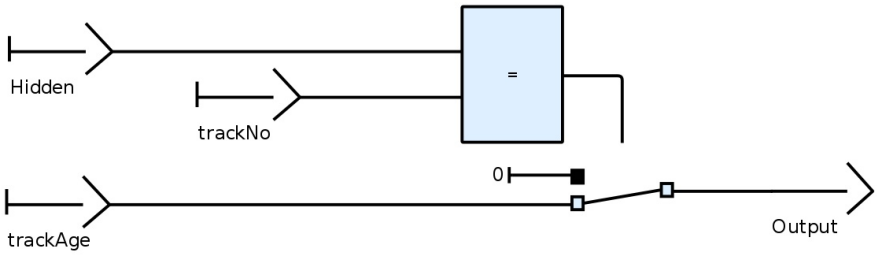
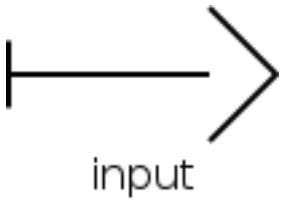
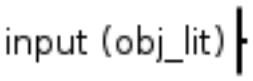


Abbildung 3.2. Operator: AgeRemoveTrack (KLighD, Original Layout)

3.2. Anpassung der Diagrammelemente

Mit der Auswertung der Annotationen wird die konkrete Darstellung des Elements bestimmt. Ist die Art des Elements vom Typ `OBJ_LIT` (object literal) handelt es sich um eine textuelle Repräsentation eines Elements oder Ausdrucks. Statt einer graphischen Darstellung durch Piktogramme werden die Informationen als Text angezeigt und mit einer minimalen Grafik (⊢) zur Verdeutlichung versehen. Für ein Eingangsdatum als `IdExpression` ist dies in den Abbildungen 3.3a (graphische Form) und 3.3b (textueller Ausdruck) dargestellt. Dabei kann es sich neben einer skalaren Zahl, einer Konstanten,

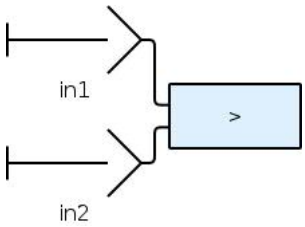


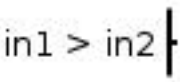


(a) Eingangswert als graphische Repräsentation **(b)** Eingangswert als textueller Ausdruck

Abbildung 3.3. Darstellungsmöglichkeiten eines Eingangssignals

Ein- und Ausgangswerten und lokalen Variablen auch um komplexere Ausdrücke handeln. In Abbildung 3.4 ist ein Beispiel eines einfachen Vergleichsoperators mit zwei Eingängen im graphischen Fall und als textueller Ausdruck dargestellt.





(a) Darstellung einer Binäroperation mit Piktogrammen

(b) Textuelle Repräsentation einer Binäroperation

Abbildung 3.4. Darstellungsmöglichkeiten einer Binäroperation in KLighD

Als weiteres grundlegendes Kriterium für die Darstellung der `entities` dient die Annotation `equation`, welche den Ausdruckstyp bestimmt. Es folgt eine Beschreibung der verfügbaren Operationen von `Expressions`, siehe dazu auch [Est10]. Die verwendete Farbgebung der Graphiken entspricht dabei immer den Farbwerten der SCADE-Modelle.

IdExpression: Beschreibt über Namen (“`id`”) referenzierte Elemente, das sind Ein- und Ausgänge, lokale Variablen und “hidden outputs”

Present: Der `Present` Ausdruck repräsentiert eine Signalauswertung.

Emission: Der `Emission` Ausruck repräsentiert eine Signalsetzung.

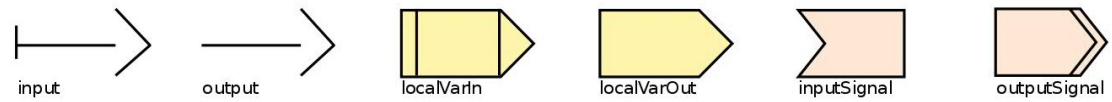
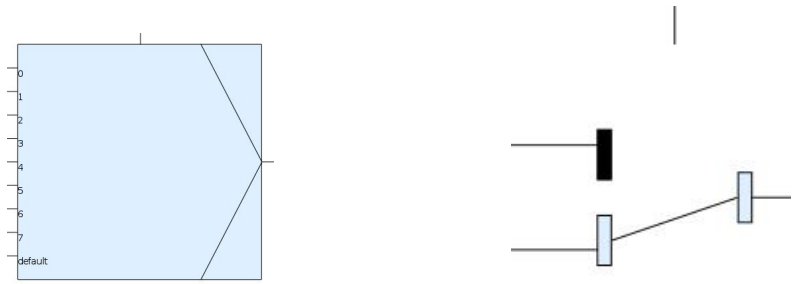


Abbildung 3.5. V.l.n.r.: Ein- und Ausgang, Variablen Lese- und Schreibzugriff, Signalauswertung und -setzung

CaseOp: Operator zur Fallunterscheidung, mit einem Steuerungseingang (`switch`) an der Nordseite, sowie beliebig viele Dateneingänge und einem Standardwert (`default`) an der Westseite des Knotens.

IfThenElseOp: Der `If-Then-Else` Operator hat einen Steuerungseingang (`if`) an der Nordseite des Knotens. An der Westseite befinden sich jeweils ein oder mehrere `then` und `else` Dateneingänge. Das schwarz eingefärbte Rechteck entspricht dabei dem `then` Dateneingang. An der Ostseite des Knotens befindet sich der ausgewählte Datenausgang.



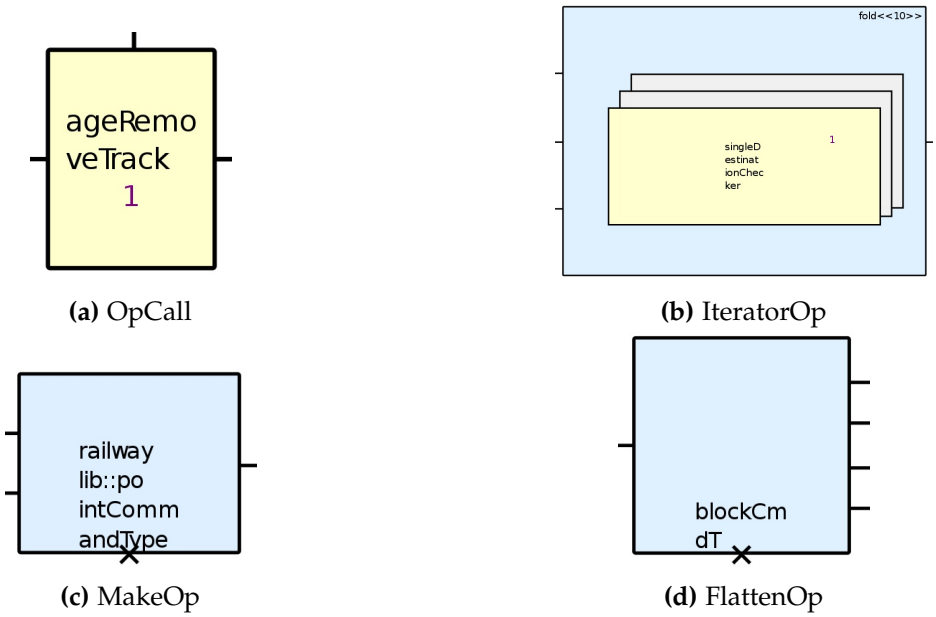
(a) Case Operator **(b)** IfThenElse Operator

Abbildung 3.6. Operatoren der Datenflusskontrolle

BinaryOp: Vergleichsoperatoren mit booleschem Ergebniswert: $<, >, \geqslant, \leqslant, <>, =$ Operator für die mathematischen Funktionen Subtraktion (binär), Division (integer und real) und modulo (integer).

ConstValue: Ein konstanter Wert, wie zum Beispiel “0” oder “true”

CallExpression: Repräsentiert den Aufruf eines Operators (`OpCall`), sowie den Aufruf höherwertiger Funktionen (beispielsweise `map` oder `fold`), welche auf den aufgerufenen Operator angewendet werden (`IteratorOp`). Repräsentiert zusätzlich die Operatoren `Make` und `Flatten`, welche mehrere Eingangsdaten zu einem Ausgangsdatum vom angegebenen Typ kombiert, beziehungsweise ein Eingangsdatum in seine Komponenten auftrennt.



(a) OpCall **(b)** IteratorOp

(c) MakeOp **(d)** FlattenOp

Abbildung 3.7. Varianten der Operatoraufrufe

ChgIthOp: Ersetzt einen Wert in einem Array durch den Wert aus einem anderen Array.

DataArrayOp: Dieser Operator kombiniert die Eingangsdaten zu einem Array.

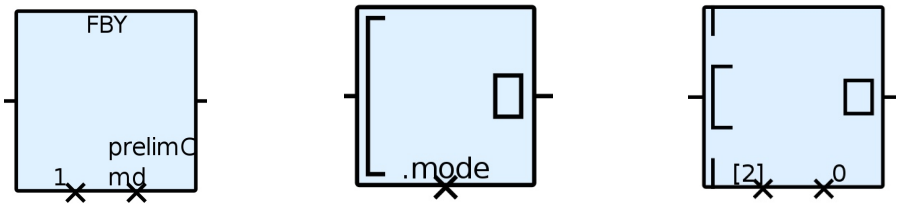
(a) ChgIth Operator **(b)** DataArray Operator

Abbildung 3.8. Array Operatoren

FByOp: Der `Followed By` Operator erzeugt den Ausgangswert von einer angegebenen Anzahl zurückliegender Zyklen des Eingangswertes. Als zusätzlichen Eingang besitzt dieser Operator einen Initialwert. Siehe dazu Abbildung 3.9a: der Wert 1 entspricht der Anzahl der Zyklen und "prelimCmd" ist der Initialwert.

PrjOp: Der Projektionsoperator extrahiert ein Element eines Arrays oder einer Struktur. In Abbildung 3.9b wird der Wert "`.mode`" der Eingangsdatenstruktur extrahiert.

PrjDynOp: Im Gegensatz zum PrjOp wird bei der dynamischen Projektion der Index des zu extrahierenden Elements nicht statisch gesetzt. Zusätzlich hat dieser Operator einen Standardwert, falls der dynamisch ermittelte Index außerhalb der Arraygrenzen liegt. In Abbildung 3.9c wird der Wert an der Stelle 2 des Eingangsarrays extrahiert. Sollte dieser Index nicht existieren, wird der Wert von Stelle 0 gewählt.



(a) `FByOp` **(b)** `PrjOp` **(c)** `PrjDynOp`

Abbildung 3.9. FollowedBy und Projektions Operatoren

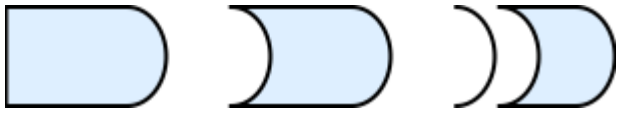
ScalarToVectorOp: Dieser Operator konstruiert ein Array mithilfe eines angegebenen Faktors, der bestimmt wie oft das Eingangsdatum für das Ausgangsdatum wiederholt werden soll. Abbildung 3.10a zeigt diesen Operator mit dem Faktor 7.

SliceOp: Extrahiert aus einem gegebenen Array ein Teilarray anhand der angegebenen Indizes. Abbildung 3.10b zeigt einen Operator, der das Teilarray von Stelle 0 bis 8 des Eingangsarrays extrahiert.

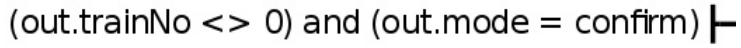
NAryOp: Operator mit beliebig vielen Eingängen, welcher zur Repräsentation der logischen Funktionen `and`, `or` und `xor` (graphische Darstellung siehe Abbildung 3.11a, als textueller Ausdruck siehe Abbildung 3.11b) verwendet wird. Des Weiteren repräsentiert er die mathematischen Funktionen Addition und Multiplikation, und dient als Operator zur Konkatenation von Arrays.

(a) `ScalarToVector` Operater **(b)** `Slice` Operator

Abbildung 3.10. ScalarToVector und Slice Operatoren



(a) Die logischen Funktionen `and`, `or` und `xor` als graphischer `NAry` Operator



(b) Ein `NAry` Operator (`and`) als textueller Ausdruck

Abbildung 3.11. Darstellungsmöglichkeiten des `NAry` Operators

UnaryOp: Eine Funktion, die auf ein einzelnes Eingangsdatum, oder einem Eingangsarray angewendet wird. Dies kann als logisches `not` auftreten (Abbildung 3.12a) oder als `Reverse` Operation, welche die Werte in einem Array permutiert, siehe dazu Abbildung 3.12b.

(a) `not` Operation **(b)** Reverse Operator

Abbildung 3.12. `Unary` Operatoren

StateMachine: Die Umrandung des `RoundedRectangle` wird mit einer gestrichelten Linie dargestellt. Der annotierte Name der Zustandsmaschine wird als Label dem Knoten hinzugefügt.

State: Der Name des Zustandes steht zentriert am oberen Rand des `RoundedRectangle` und wird durch eine einfache Linie vom zugehörigen Datenfluss-

diagramm abgetrennt. Der Startzustand einer Zustandsmaschine wird durch eine standardmäßig um das dreifach erhöhte Linienstärke hervorgehoben. Abbildung 3.13 zeigt einen Ausschnitt eines mit KLighD erzeugten Zustandsgraphen.

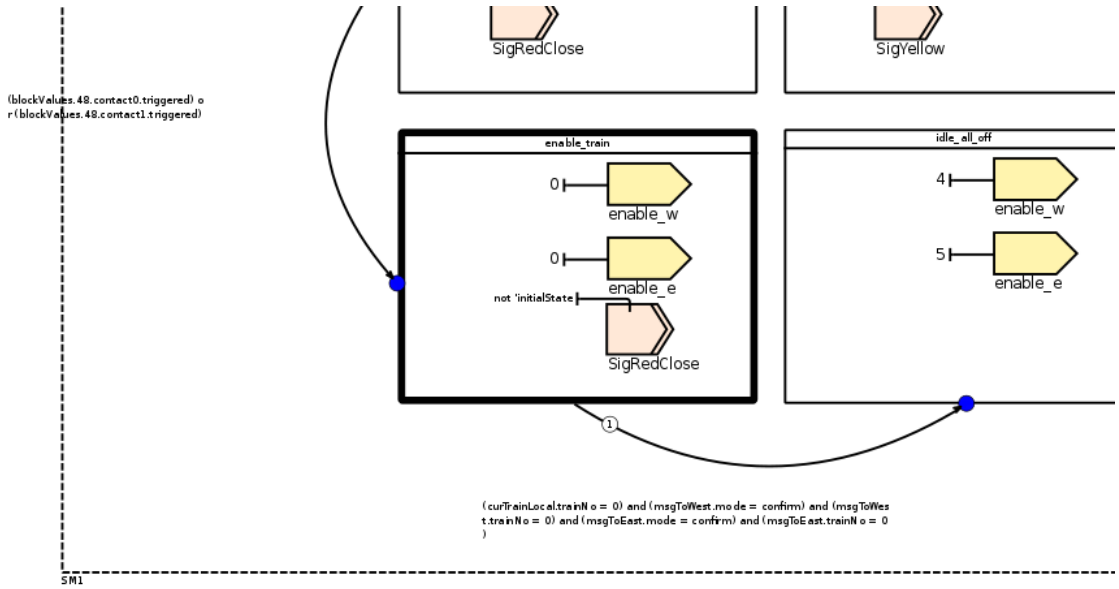


Abbildung 3.13. Ausschnitt eines Zustandsgraphen in KLighD

Zustandsgraphen können auch ineinander verschachtelt sein. Das bedeutet, dass sich in einem Zustand ein oder mehrere weitere Zustandsgraphen befinden können (siehe Abbildung 3.14). Diese haben wiederum Zustände und Datenflussdiagramme. Die verschachtelten Zustandsgraphen müssen dabei nicht zwingend im gleichen KAOM-Modell gespeichert sein. Ist ein Zustand leer, so liegen die Daten aufgrund der Übersichtlichkeit der Modelle in einem separaten Modell. Die Visualisierung erfolgt nach den oben beschriebenen Regeln.

Kanten und Transitionen: Die Darstellung von Kanten erfolgt mit einer einfachen Linie. Im Gegensatz dazu werden Transitionen am Endpunkt immer mit einem Pfeil gekennzeichnet. Die Prioritäten der Transitionen werden durch Nummern in der Nähe des Startpunktes angezeigt. Die farbliche Kennzeichnung am Endpunkt einer Transition bestimmt den Transitionstyp. Diese zusätzlichen graphischen Details einer Transition sind in Abbildung 3.14 zu sehen.

Die Positions- und Größenangaben aller Elemente in den für diese Studienarbeit ver-

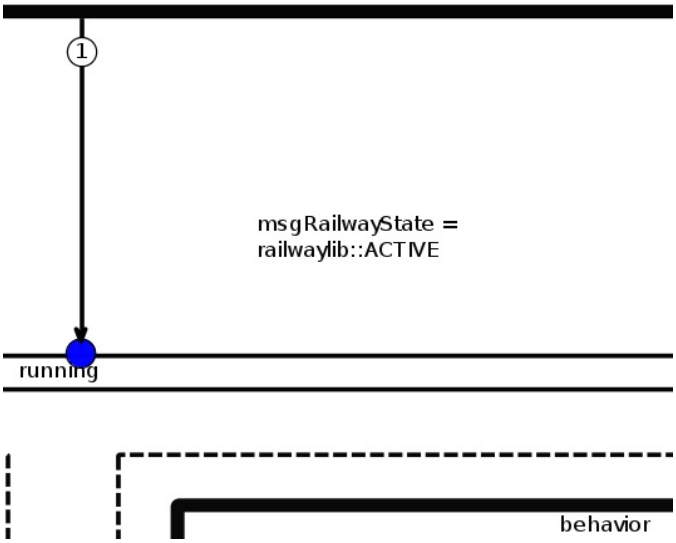


Abbildung 3.14. Ausschnitt eines verschachtelten Zustandsgraphen (KLighD)

wendeten Testmodellen im SCADE-Format sind mit einem Skalierungsfaktor der Größe 10 versehen. Beim Import der SCADE-Modelle werden die konkret gespeicherten Werte in das KAOM-Format übernommen. Bei der Diagrammsynthese werden diese Werte wieder zurück skaliert.

3.3. Das Diagrammlayout

Mithilfe der originalen Positionsangaben aller Diagrammelemente lässt sich ein originalgetreues Layout erstellen. Alternativ kann das Layout der Diagramme mit einem automatischen Algorithmus (`Klay Layered`) erfolgen.

Bei Verwendung des automatischen Layouts ist auf die Layoutoption `FixedOrder` bei den `ports` zu achten. Dies ist notwendig, damit zum Beispiel bei Vergleichsoperatoren immer der erste `port`, also das erste Eingangsdatum, auch an die erste Stelle gesetzt wird. Nur so ist gewährleistet, dass die Lesbarkeit der Vergleichsoperatoren immer gleich und korrekt bleibt. Wird bei der Darstellung das originale Layout ausgewählt, so spielt das Festlegen der Seite für jeden `port` keine Rolle mehr, da sich Start-, Ziel- und alle Knickpunkte der Kante als Positionsangaben in den Annotationen finden und übernommen werden.

Kanten werden als Polylinien angezeigt. Das `EdgeRouting` für Kanten erfolgt orthogonal, was den Lesefluss der Diagramme unterstützt. Transitionen werden im Gegenzug als Splines dargestellt, was die Unterscheidung von einfachen Kanten zusätzlich hervor-

hebt. Die annotierten Informationen der Knickpunkte werden nur bei der Darstellung des Originallayouts verwendet um den originalen Verlauf der Kanten und Transitionen anzuzeigen. Bei der automatischen Layoutgenerierung wird auf diese verzichtet, lediglich die festgelegten Start- und Zielports werden verwendet. Hierarchieübergreifende Kanten und Transitionen werden nicht unterstützt.

3.4. Darstellungsoptionen

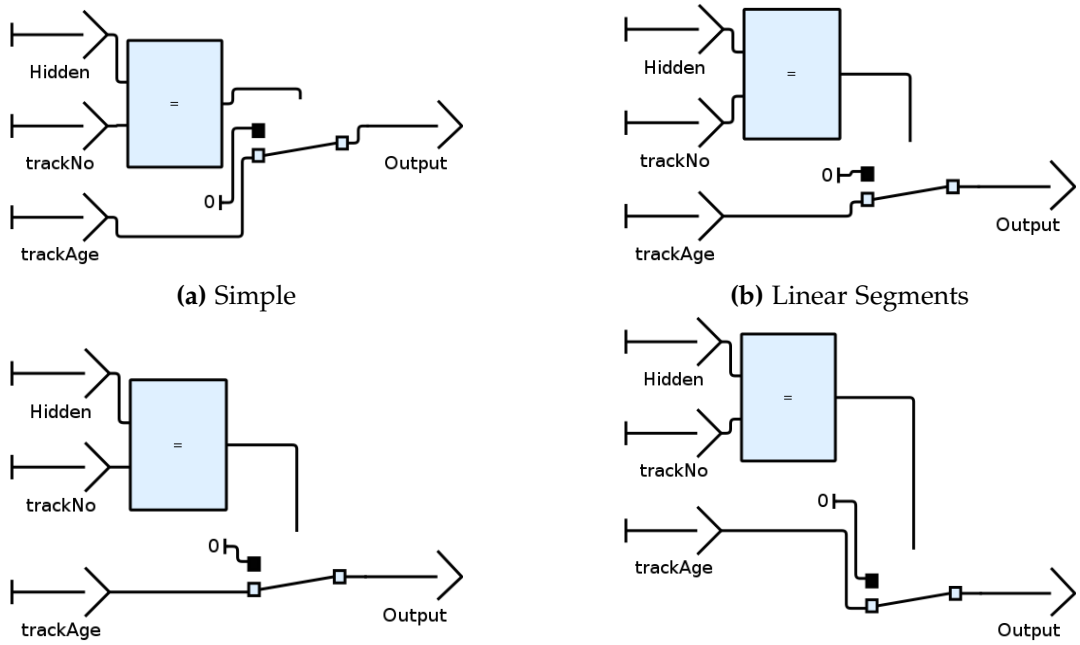
Ein Teilziel dieser Studienarbeit für den Visualisierungsteil ist die Implementierung verschiedener Darstellungsoptionen. Im folgenden werden die umgesetzten Optionen erläutert. Eine Erweiterung oder Anpassung dieser Optionen ist problemlos möglich.

Layout Die wesentliche Darstellungsoption ist die Möglichkeit zwischen dem originalen Layout des SCADE-Operators und einem automatischen Layout zu wechseln. Für das originale Layout werden die originalen Größen- und Positionsangaben aller Knoten verwendet. Bei den Kanten werden alle originalen Positionsangaben der Start-, Ziel- und Knickpunkte verwendet, sowie die Positionsangabe des Labels bei den Transitionen. Entscheidet man sich für ein automatisches Layout, bietet der verwendete Layoutalgorithmus vier vorhandene Strategien zur Knotenplatzierung an, siehe Tabelle 3.1. Die Anwendung dieser vier Strategien ist in der Abbildung 3.15) dargestellt.

Tabelle 3.1. Automatische Knotenplatzierungsstrategien in KIELER

Knotenplatzierung	Beschreibung
Simple	Ein sehr einfacher und sehr schneller Algorithmus zur vertikal zentrierten Platzierung aller Knoten.
Linear Segments	Ein Algorithmus, der mit Hilfe linearer Segmente lange Kanten ausrichtet. Die Knoten werden entsprechend der *Pendulum* Methode ausgerichtet. Diese ist ähnlich der baryzentrischen Methode zur Knotenanordnung.
Brandes Koepf	Dieser Algorithmus platziert die Knoten in Blöcken so, dass sich möglichst gerade Kanten ergeben.
Buchheim Juenger Leipert	Ein Algorithmus der die Knoten in Klassen gruppiert, mit möglichst geraden Kanten als Ziel.

Rotation von Elementen (automatisches Layout) Mit dieser Diagrammoption wird die Rotation von Elementen gemäß ihres annotierten Rotationswertes im automatischen



(a) Simple

(b) Linear Segments

(c) Brandes Koepf

(d) Buchheim Juenger Leipert

Abbildung 3.15. Operator "AgeRemoveTrack", KLighD Darstellung mit automatischem Layout

Layout aktiviert. Dies ermöglicht in Kombination mit den verschiedenen Knotenplatzierungsstrategien ein vielfältigeres Testszenario für den Vergleich von automatischem und originalgetreuen Layout. Dabei ist zu beachten, dass nur Rotationen um einen Winkel eines Vielfachen von 90° unterstützt werden, da die `ports` der ein- und ausgehenden Kanten nur auf diese Werte gesetzt werden können.

Transitionslabel Diese Diagrammoption ermöglicht das Ein- und Ausblenden der Transitionslabel (beinhaltet die Bedingung zum Wechsel des Zustandes, sowie die emittierten Signale).

Kantendarstellung Als Diagrammoption lassen sich die Kanten mit Pfeilen am Ende darstellen. Beim automatischen Layout werden alle Eingänge auf die West- oder Nordseite eines Knotens gelegt und die Ausgänge auf die Ostseite; im originalen Layout muss dies aber nicht immer der Fall sein. Somit ist im Originallayout das Anzeigen der Datenflussrichtung mittels der Pfeile an den Kanten sehr sinnvoll.

Spacing Das `LayoutSpacing` ermöglicht den Abstand zwischen den Knotenelementen zu vergrößern oder zu verkleinern.

Schriftgröße Bestimmt die Schriftgröße aller Ein- und Ausgänge, lokalen Variablen und Signalen. Weitere Textelemente sind nicht mit inbegriffen um die Lesbarkeit bei sehr langen Operatornamen oder verschachtelten textuellen Ausdrücken nicht unnötig zu verschlechtern, in dem die Schrift andere Elemente überlagert.

Linienstärke Ebenfalls ist es möglich die Linienstärke der gezeichneten Kanten anzupassen. Lässt man sich sehr große Modelle anzeigen, werden die Linien in KLighD etwas schwächer dargestellt, was mit dieser Darstellungsoption kompensiert werden kann. Auch für eine Beamerpräsentation oder eine Druckversion der Diagramme ist die Anpassung der Linienstärke sehr sinnvoll.

Kapitel 4

Zusammenfassung und Ausblick

Im letzten Abschnitt dieser Studienarbeit wird zunächst eine Zusammenfassung der erreichten Ziele gegeben. Es wird dabei auch auf Probleme und Einschränkungen der Umsetzung eingeganen. Abschließend wird ein Ausblick über mögliche weiterführende Arbeiten gegeben.

4.1. Zusammenfassung

Es ist gelungen ein Eclipse-Plugin zu implementieren, dass SCADE-Modelle lädt und die Daten in das KAOM-Format migriert. Dabei können die Modelle Datenflussdiagramme oder Zustandsdiagramme sein. Die migrierten Modelle lassen sich anschließend mit KLighD visualisieren. Bei der Visualisierung kann zwischen dem originalgetreuen Layout und einem automatischen Layout gewechselt werden. Auch weitere Darstellungsoptionen, wie Linienstärke und Schriftgröße lassen sich anpassen. Diese Optionen lassen sich dabei mit wenig Programmieraufwand erweitern, beziehungsweise neue Optionen können hinzugefügt werden.

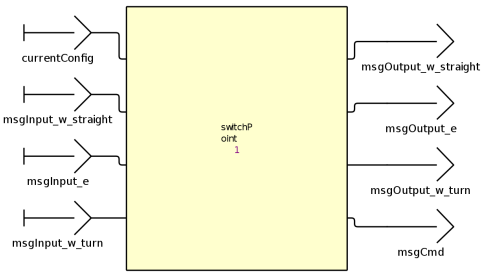
Ein paar komplexere Beispieldiagramme unter Verwendung verschiedener Darstellungsoptionen sind in Anhang A zu finden.

Aufgrund der Skalierung der Elemente im SCADE-Format und der anschließenden Kompensation dieser Skalierung während der Diagrammsynthese kann es zu leichten Ungenauigkeiten beim Zeichnen von beispielsweise Kanten kommen. Die notwendige Typenkonvertierung von `integer` zu `float` der Positionsangaben kann ebenfalls eine Ursache dieser Ungenauigkeiten sein.

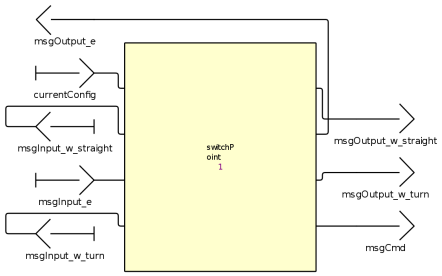
Die Darstellung von `hidden inputs` bei Operatoraufrufen ist nur teilweise implementiert. Sie funktioniert für eine feste Anzahl dieser Eingangswerte bei vordefinierten SCADE-Operatoren, wird bei der Darstellung von eigens erstellten Operatoren aber noch nicht unterstützt.

Bei der aktivierten Rotation von Elementen für das automatische Layout ist folgendes zu beachten. Das Element wird korrekt rotiert und entsprechend auch die zugehörigen

ports. Die ports von Elementen, die mit dem rotierten Element verbunden sind, bleiben aber weiterhin auf ihrer Standardseite festgesetzt. Somit kann diese Diagrammoption in einigen Fällen nicht zur Lesbarkeit der Modelle beitragen. In Abbildung 4.1a sieht man den Operator "SwitchPoint-LeftMerge" mit automatischem Layout, welcher vier Eingänge auf der Westseite und vier Ausgänge auf der Ostseite besitzt. In Abbildung 4.1b wurde die Rotationsoption aktiviert, und man sieht, dass zwei Eingänge und ein Ausgang rotiert werden. Die ports der Anbindung an den Operaotraufruf bleiben unberührt, so dass in diesem Fall keine Verbesserung der Lesbarkeit erzielt wurde. Für



(a) Darstellung des Operators ohne Rotation



(b) Darstellung des Operators mit aktivierter Rotation

Abbildung 4.1. Automatisches Layout des Operators "SwitchPoint-LeftMerge"

noch ausführlichere Tests der automatischen Knotenplazierungsstrategien wäre es an dieser Stelle denkbar, auf die Festsetzung jeglicher ports zu verzichten. Der Nachteil ohne diese Festsetzung ist, dass keine Verallgemeinerungen über den Datenfluss in den Diagrammen mehr erzielt werden, wodurch die Verständlichkeit und der Lesefluss beeinträchtigt werden. Alternativ wäre es auch möglich die ports, mithilfe der Start- und Zielpunkte der entsprechenden Kanten, fest zu positionieren.

Der Textumbruch von Operatornamen und Labels trägt teilweise nicht zur besseren Lesbarkeit der Modelle bei. Für das automatische Layout sind lange Textfelder kein Problem, da die Platzierung der Knoten dann ebenfalls automatisch angepasst wird. Bei der Anzeige des originalen Layouts kann eine zu große Expansion der Knoten allerdings zur Überlagerung von Elementen führen (darauf wird im nachfolgenden Unterkapitel noch genauer eingegangen).

4.2. Ausblick

Im Hinblick auf einen weiteren Vergleich zwischen originalem Layout und automatischem Layout könnte man auf die Verwendung von doppelten Elementen in den Modellen verzichten. In Abbildung 4.2 ist ein Operator dargestellt, der zweimal das gleiche graphische Element (`in_localTrainNumberT`) enthält. Die abstrakten Daten sind

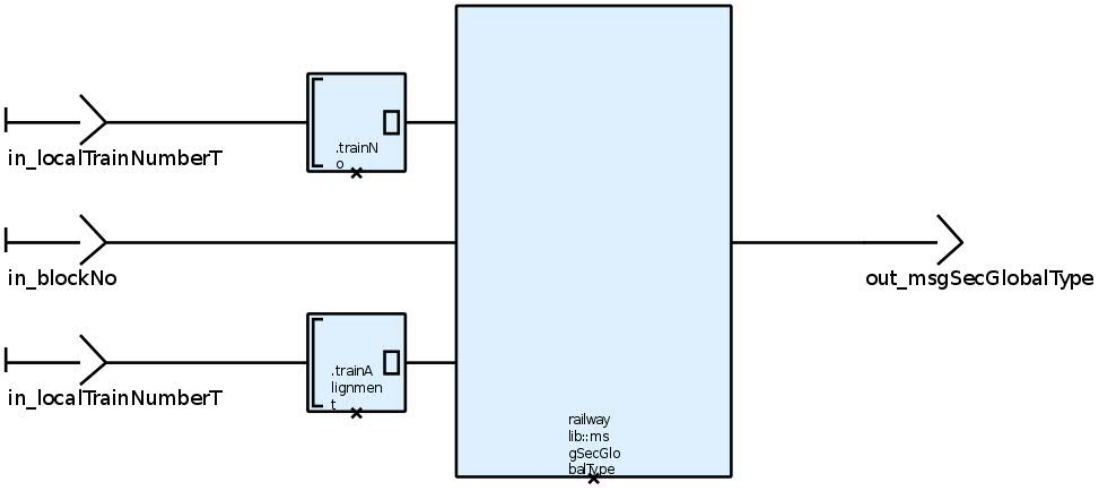


Abbildung 4.2. Operator mit doppeltem Element

selbstverständlich nur einmal vorhanden, die Verwendung der doppelten graphischen Elemente dient dabei nur der Lesbarkeit. Für das automatische Layout könnte man an dieser Stelle aber auch auf das doppelte Element verzichten. Gerade bei größeren Modellen wäre ein Vergleich der verschiedenen Knotenplatzierungsstrategien mit dem originalen Layout sehr interessant.

Eine weitere nützliche Funktion wäre die Expansion von Operatoraufrufen. Klickt man einen `OpCall` an, so wird das entsprechende zugehörige Modell geladen und an Stelle der “Blackbox” angezeigt.

Ein großes Problem ist die ästhetische Platzierung von Textfeldern, so dass diese zum einen gut lesbar sind, aber auch das Layout nicht überladen oder gar zerstören. In Abbildung 4.3 sieht man, dass durch den langen Namen des Operatoraufrufs, der Knoten expandiert wurde. Zu sehen ist dies dadurch, dass beim originalen Layout die Kanten nicht mehr korrekt an der Umrandung anliegen, da deren Positionen genau festgelegt sind. Zusätzlich kann es zur Überlagerung von Modellelementen kommen, was die Lesbarkeit deutlich vermindert. Dies ist in der Abbildung zum Beispiel daran

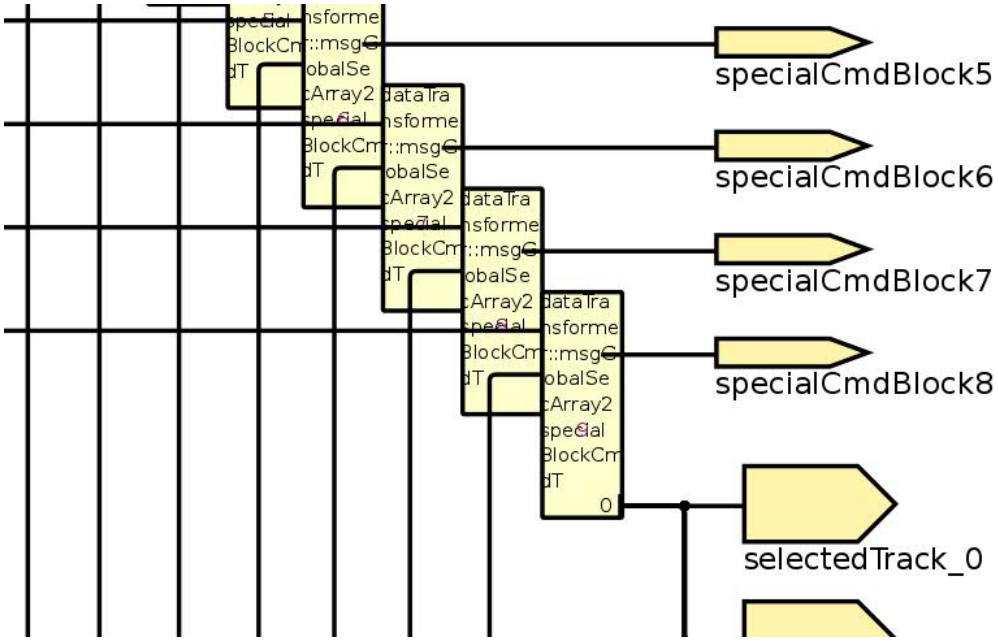


Abbildung 4.3. Darstellungsprobleme bei langen Textfeldern

zu erkennen, dass ein konstantes Eingangsdatum mit dem Wert 0 scheinbar innerhalb des Operatoraufrufes liegt.

Als letzte weiterführende Arbeit sei die erneute Migration der Modelle erwähnt. Beispielsweise in das Ptolemy-Format.[1] Mit dem dazugehörigen Simulator könnte man diese SCADE-Modelle dann auch simulieren.

[1] Ptolemy II project: `http://ptolemy.eecs.berkeley.edu/`

Literaturverzeichnis

[Ber07] Berry, Gérard: *SCADE: Synchronous Design and Validation of Embedded Control Software*. In: Ramesh, S. und Prahladavaradan Sampath (Herausgeber): *Next Generation Design and Verification Methodologies for Distributed Embedded Control Systems*, Seiten 19–33. Springer Netherlands, 2007, ISBN 978-1-4020-6253-7. http://dx.doi.org/10.1007/978-1-4020-6254-4_2.

[Est10] EsterelTechnologies: *SCADE Suite Design Notations Quick Reference*. http://www.esterel-technologies.com/wp-content/uploads/2013/02/SCADE-Reference-card.pdf, 2010. [Online; accessed 11.06.2014].

[FvH10] Fuhrmann, Hauke und Reinhard von Hanxleden: *Taming Graphical Modeling*. In: *Proceedings of the ACM/IEEE 13th International Conference on Model Driven Engineering Languages and Systems (MoDELS'10)*, Band 6394 der Reihe *LNCS*, Seiten 196–210. Springer, October 2010.

[FvHK+14] Frey, Patrick, Reinhard von Hanxleden, Christoph Krüger, Ulf Rüegg, Christian Schneider und Miro Spönemann: *Efficient Exploration of Complex Data Flow Models*. In: *Proceedings of Modellierung 2014*, Vienna, Austria, March 2014.

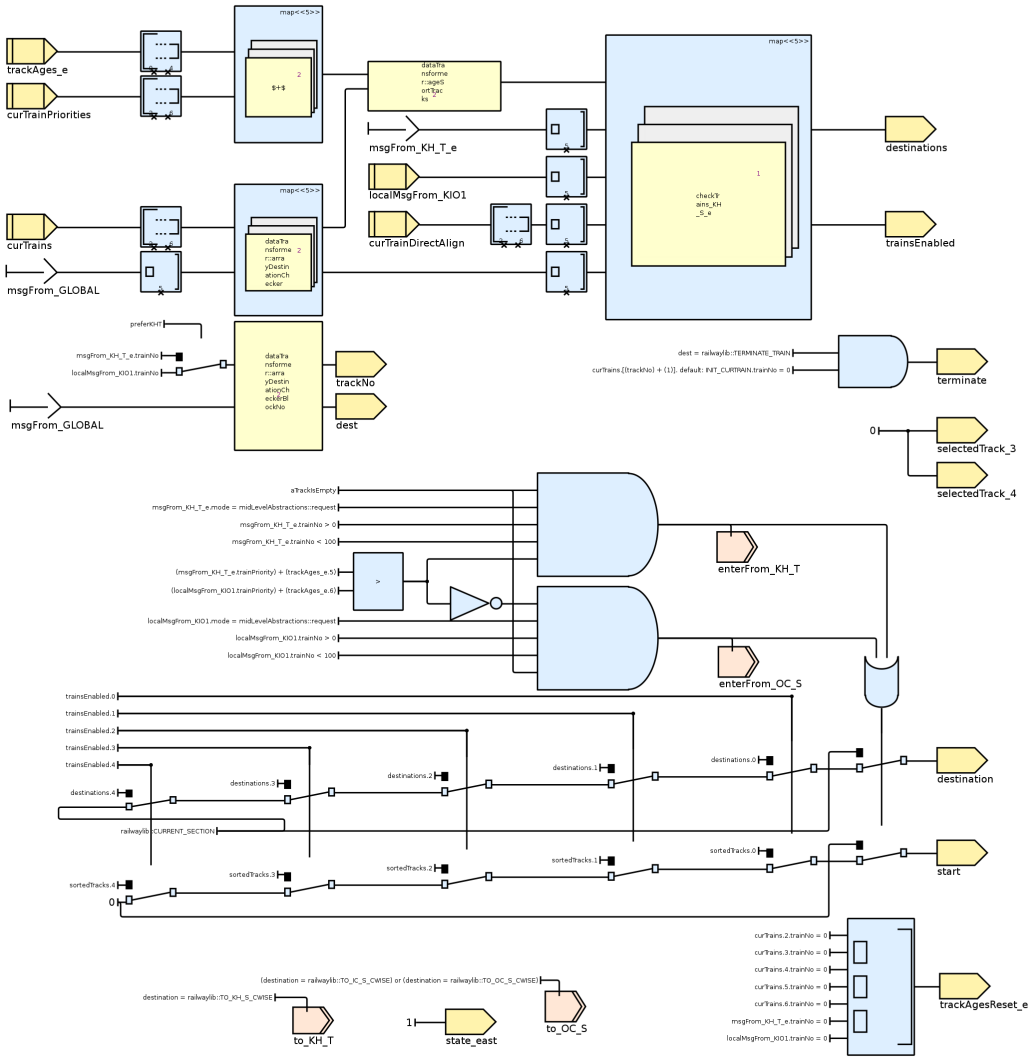
[SSvH12] Schneider, Christian, Miro Spönemann und Reinhard von Hanxleden: *Transient View Generation in Eclipse*. In: *Proceedings of the First Workshop on Academics Modeling with Eclipse*, Kgs. Lyngby, Denmark, Juli 2012.

[SSvH13] Schneider, Christian, Miro Spönemann und Reinhard von Hanxleden: *Just Model! – Putting Automatic Synthesis of Node-Link-Diagrams into Practice*. In: *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'13)*, San Jose, CA, USA, 15–19 9 2013.

[SSvH14] Schulze, Christoph Daniel, Miro Spönemann und Reinhard von Hanxleden: *Drawing Layered Graphs with Port Constraints*. Journal of Visual Languages and Computing, Special Issue on Diagram Aesthetics and Layout, 25(2):89–106, 2014, ISSN 1045-926X.

Anhang A

Beispieldiagramme



Diagrammausschnitt: “Kicking Horse Station” dargestellt mit KLighD, originales Layout

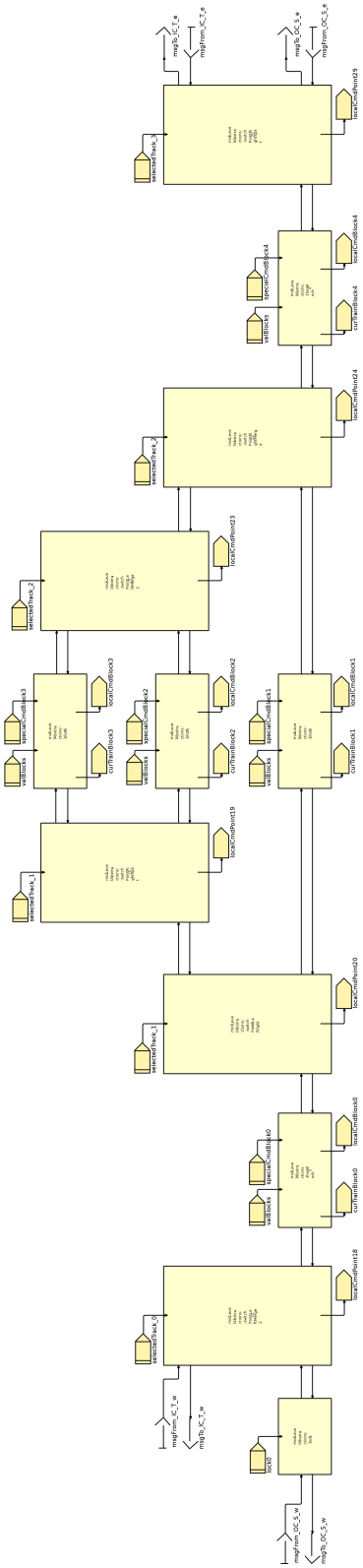


Diagramm: “Inner Circle Station Data Flow” dargestellt mit KLighD, originales Layout, Kantendarstellung mit Pfeilen

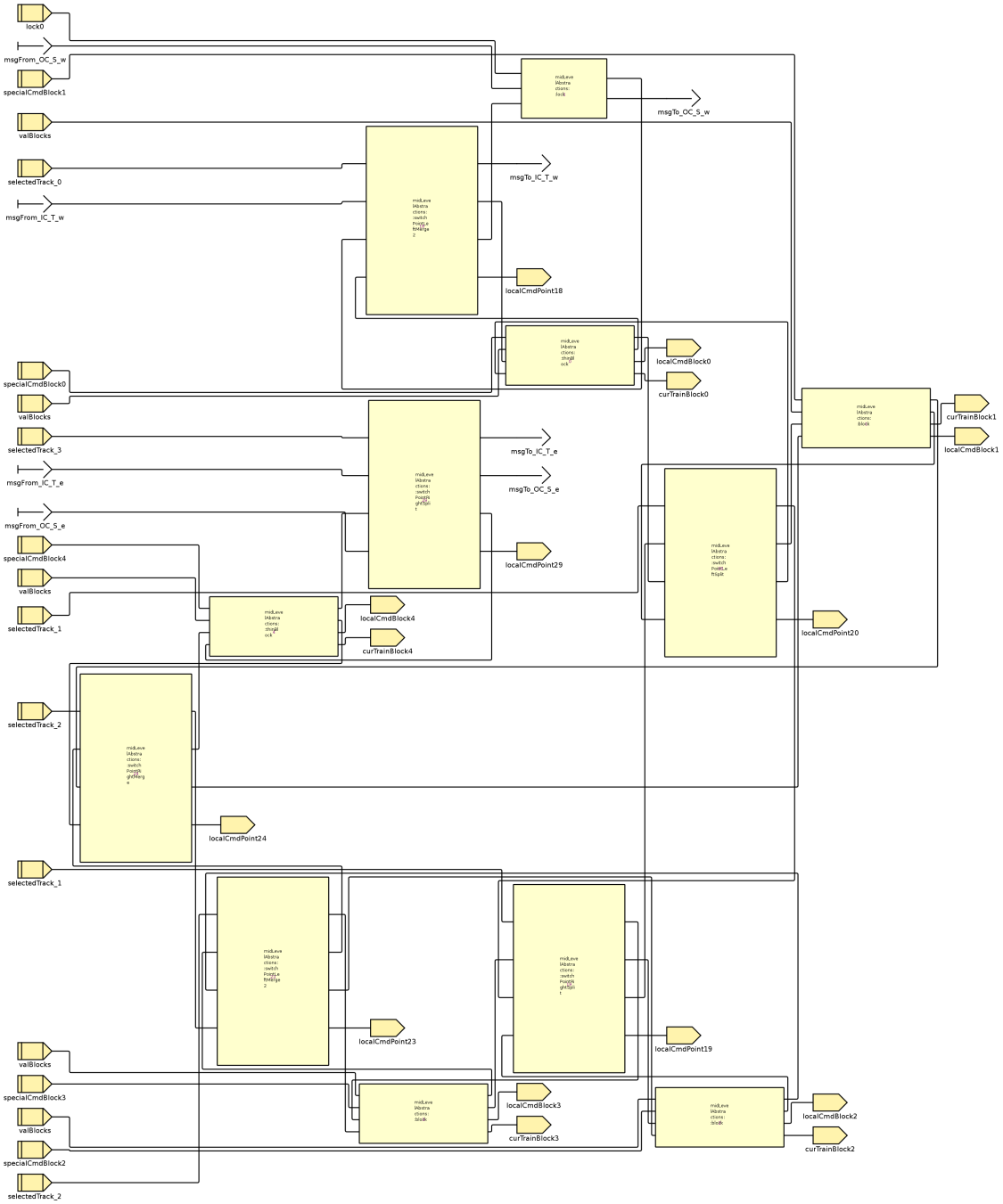


Diagramm: “Inner Circle Station Data Flow” dargestellt mit KLighD, automatisches Layout, Knotenplatzierung: Linear Segments

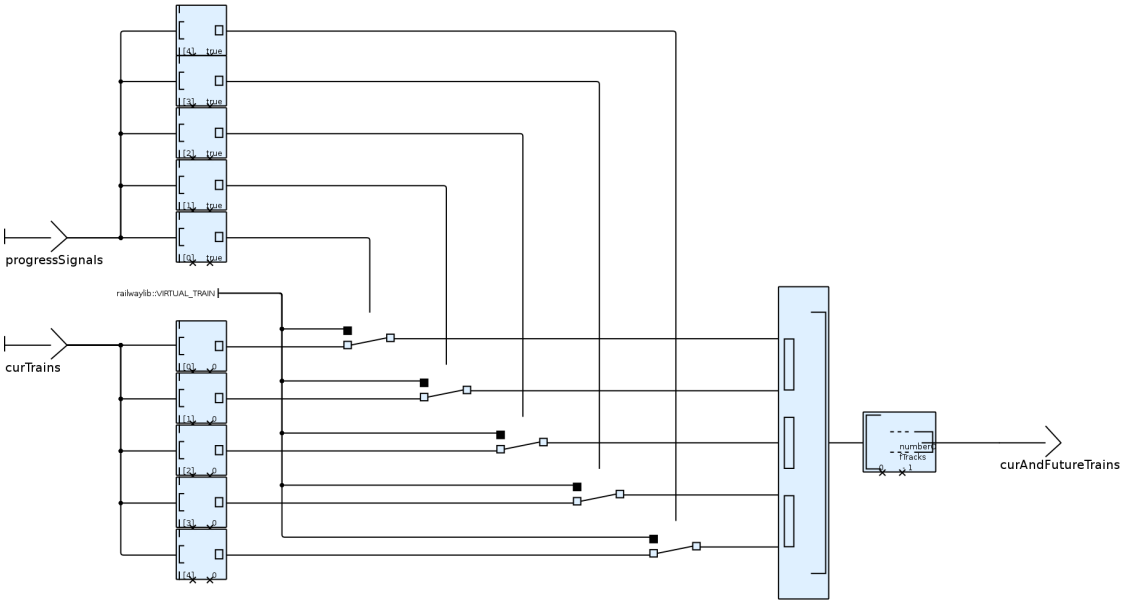


Diagramm: “Include Progressed Tracks” dargestellt mit KLighD, original Layout

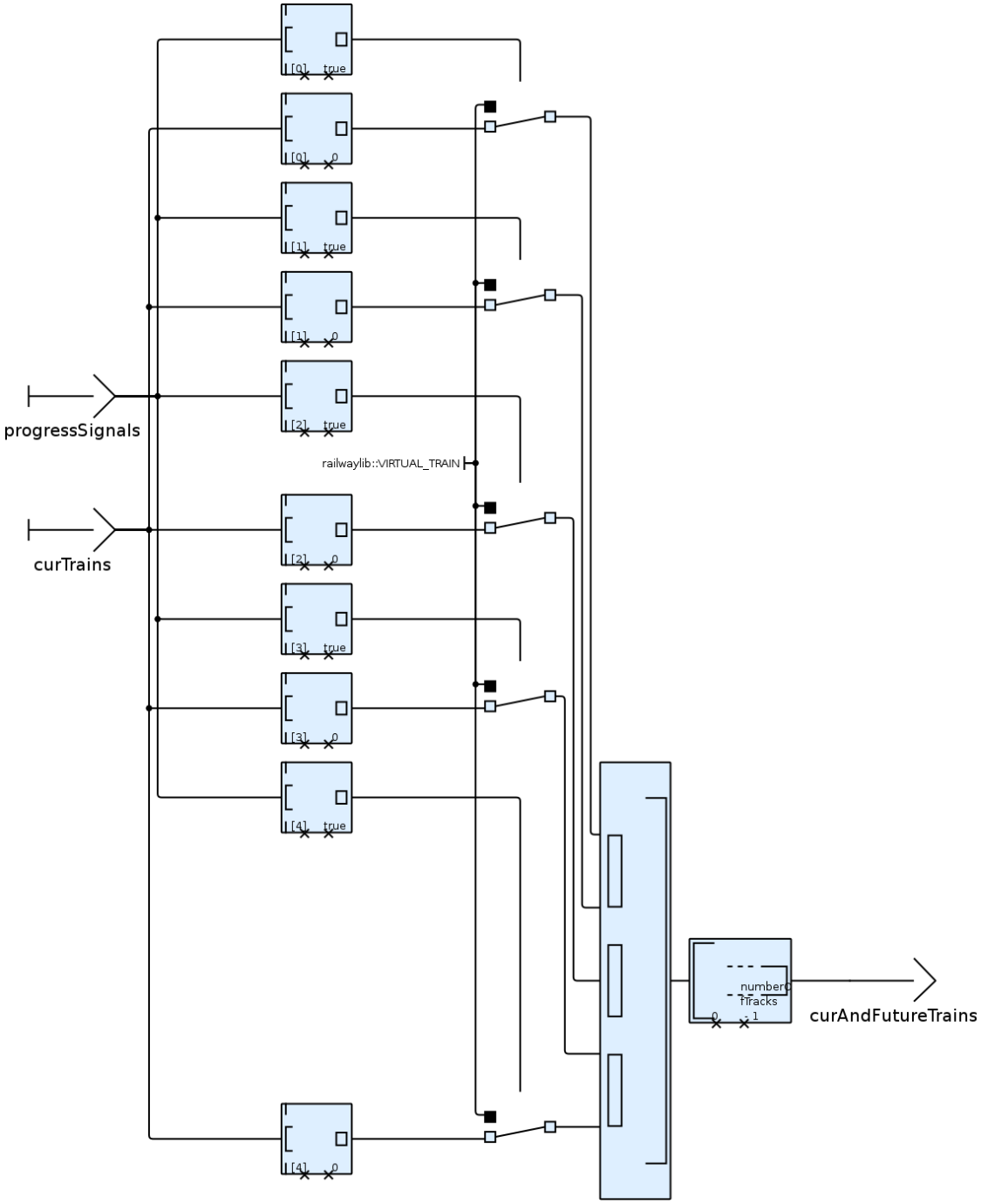


Diagramm: “Include Progressed Tracks” dargestellt mit KLighD, automatisches Layout, Knotenplatzierung: Brandes Koepf