Arbeiten mit Reaktor

Seminararbeit

221300a Ton Seminar

Audiovisuelle Medien

Hochschule der Medien, Stuttgart
Fakultät Electronic Media

Eingereicht am: 15. Februar 2018

Prüfer: Prof. Oliver Curdt

Zusammenfassung

Die vorliegende Arbeit ist eine schriftliche Ausarbeitung des im Rahmen des Ton Seminars gehaltenen Vortrags *"Arbeiten mit Reaktor"*. Sie erläutert die Benutzung von Reaktor, einer visuellen Programmierumgebung zur Entwicklung und Anwendung von Software-Instrumenten und virtuellen Effektgeräten der *Native Instruments GmbH*. Zunächst wird anhand der Benutzeroberfläche, der Projekthierarchie und dem `Snapshot`-System die allgemeine Programmstruktur beschrieben. Diese Grundlagen befähigen zur einfachen Anwendung bereits existierender Instrumente und Effekte. Im Anschluss daran, werden die `Primary`- und die `Core`-Ebene und die damit verbundenen Konzepte wie deren Signaltypen und die Fehlerdiagnose erläutert. Das Verständnis dieser Entwicklungsschichten und deren Eigenschaften und Unterschiede bilden die Basis für das Umsetzen eigener Projekte in Reaktor.

Inhaltsverzeichnis

1 Einführung **1**

1.1 Reaktor . . . 1
1.2 Benutzeroberfläche . . . 1
1.3 Dateihierarchie . . . 2
1.4 Snapshots und Presets . . . 3

2 Die Primary-Ebene **4**

2.1 Signaltypen . . . 4
2.1.1 Audio . . . 4
2.1.2 Event . . . 5
2.1.3 Table Reference . . . 5
2.2 Fehlerdiagnose . . . 5
2.2.1 Event Watcher, Audio Core Event Watcher . . . 6
2.3 Init und Re-Init . . . 7
2.4 Send und Receive, Internal Connections . . . 7
2.5 for-Schleife . . . 8
2.6 Kontrollstruktur . . . 8
2.7 Arrays (Audio-/Event-Table) . . . 9

3 Die Core-Ebene **10**

3.1 Signaltypen . . . 10
3.1.1 Skalare . . . 10
3.1.2 Object Bus Connections . . . 11
3.1.3 Boolean Control . . . 11
3.1.4 Bundles . . . 12
3.1.5 Scoped Buses . . . 12
3.2 Debugging . . . 12
3.3 Initialisierung . . . 13
3.4 for-Schleife . . . 13
3.5 Kontrollstruktur . . . 14
3.6 Arrays und Tables . . . 14

4 Fazit **15**

Literaturverzeichnis **16**

Abbildungsverzeichnis **17**

1 Einführung

1.1 Reaktor

Reaktor ist eine visuelle Programmierumgebung zur Entwicklung und Anwendung von Software-Instrumenten und virtuellen Effektgeräten der *Native Instruments GmbH*. Durch das Hinzufügen und Verbinden von signalerzeugenden und signalbearbeitenden Modulen, lassen sich Datenströme generieren und steuern. Dies ermöglicht das Erzeugen und Manipulieren von Audiomaterial, das Reagieren auf Benutzereingaben des Anwenders samt visuellem Feedback auf der Panelebene und das Verwalten von Arraydaten.

Reaktor kann sowohl Stand-Alone betrieben, als auch in einer Digital Audio Workstation als Plug-In verwendet werden. Unter dem Namen *Reaktor Player* ist eine im Funktionsumfang eingeschränkte Version von Reaktor, die lediglich zum Benutzen von Reaktorensembles verwendet werden kann, erhältlich.

Neben einer Factory Library, die eine Vielzahl an Instrumenten, Effekten und Makros enthält, gibt es im Internet eine User Library mit von anderen Reaktornutzern erstellten Projekten. Beide Quellen sind größtenteils Open Source und somit eine hilfreiche Lernressource.

1.2 Benutzeroberfläche

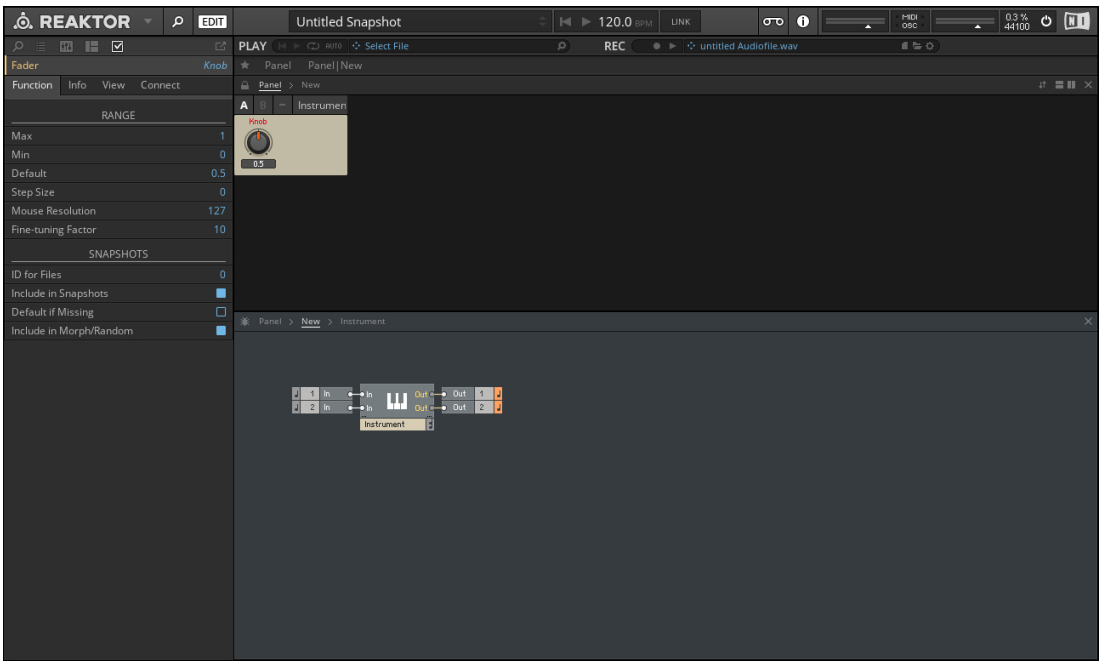

Abbildung 1: Benutzeroberfläche von Reaktor 6

Die Benutzeroberfläche von Reaktor ist in drei Bereiche unterteilt: eine Werkzeugleiste, eine Seitenleiste und einen teilbaren Hauptbereich. In der Werkzeugleiste können Anwendungseinstellungen wie die Samplingrate oder das Projekttempo vorgenommen, zwischen dem Play- und Edit-Modus gewechselt und die Eingangs- und Ausgangspegel abgelesen werden. Im Seitenbereich befinden sich ein Dateibrowser und Presetbrowser zur Auswahl von Projektdateien, Modulen, Makros und Snapshots, ein Reiter zur Verwaltung von MIDI- und OSC-Controller-Einstellungen, einer zur Verwaltung von Panelsets und ein Eigenschaftsreiter, der zum Konfigurieren des in der Projektstruktur zuletzt ausgewählten Elements dient. Panelsets werden die Anordnung von Instrumenten im Hauptbereich genannt. In diesem hat man Zugriff auf die Projektstruktur und das Panel mit den entsprechenden Bedienelementen.

1.3 Dateihierarchie

Nachstehend ist die Hierarchie einer mit Reaktor erstellten Projektdatei dargestellt. Vergleicht man ihre Struktur mit einem Modularen Synthesizer, so entspricht das Ensemble dem Rack, die Instrumente den Modulen und die Makros bzw. Module den Schaltungen bzw. Bauteilen. Entsprechend der verschiedenen Dateiendungen lassen sich die einzelnen Bestandteile gesondert speichern und laden. Möchte man beispielsweise in der Core-Ebene programmieren, muss ein neues Ensemble erstellt werden, dass mindestens ein Instrument enthält, in dem wiederum eine `Core Cell` eingefügt wurde. Diese `Core Cell` stellt dann die Schnittstelle zwischen der Primary- und der Core-Ebene dar.

- Ensemble = *Projektdatei (.ens)*
    - Instrumente = *Synthesizer, Sampler, Sequencer, Effekt (.ism)*

      Primary-Ebene
      - Macros *(.mdl)*
          - Module
      - Core Cells *(.rcc)*

        Core-Ebene
        - Macros *(.rcm)*
            - Module

1.4 Snapshots und Presets

In Reaktor wird zum Speichern von Projektzuständen zwischen Snapshots und Presets unterschieden. Snapshots speichern die Zustände einzelner Instrument, wobei Ensemble-Snapshots, die Snapshots aller eingefügter Instrumente verwalten. Insgesamt ist die Anzahl an Snapshots in einem Ensemble auf 2048 begrenzt. Des weiteren kann man zwischen den Werten zweier Snapshots linear interpolieren und in Abhängigkeit der Werte als Grenzwerte zufällige Snapshotwerte generieren. Snapshots erlauben außerdem das Wechseln per MIDI Programm Change Messages. Sie lassen sich nur im Edit-Modus erstellen und sind im Ensemble eingebettet (exportierte Snapshotbänke ausgeschlossen). Presets hingegen speichern den Gesamtzustand eines Ensembles. Sie können nur gespeichert und geladen werden und werden im Play-Modus als zusätzliche Datei gespeichert. Für Benutzer von Reaktor, die mit der Vollversion arbeiten und nicht auf die Anwender der Reaktor Player Version abzielen, bieten Presets deshalb keine nennenswerte Vorteile.

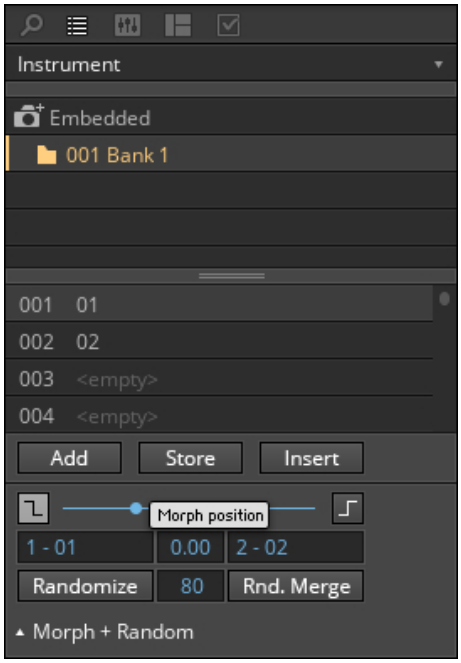

Abbildung 2: Seitenleistenreiter zum Erstellen und Bearbeiten von Snapshots mit Zufalls- und Interpolationsfunktionen

2 Die Primary-Ebene

Dieser Abschnitt der Seminararbeit erläutert die Eigenschaften der Primary-Ebene. Sie ist die oberste Entwicklungsschicht in Reaktor und dient überwiegend zum Erstellen und Verbinden von höher abstrahierten und optimierten Modulen wie Oszillatoren, Filtern und Hüllkurven oder den entsprechenden Makros mit integrierter Core Cell. Zudem wird die Benutzeroberfläche über die Primary-Ebene erstellt und die Interaktion des Anwenders mit den Instrumenten reagiert.

2.1 Signaltypen

In der Primary-Ebene wird zwischen Audio-, Event- und Table Reference-Signalen unterschieden. Für den jeweiligen Signaltyp existieren entsprechende Verbindungen, die durch unterschiedlich eingefärbte Modulports visualisiert werden.

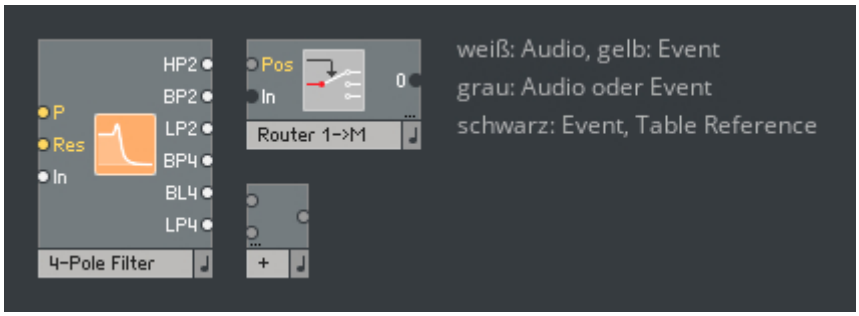

Abbildung 3: Signaltypen der Primary-Ebene

2.1.1 Audio

Alle Vorgänge die in Reaktor mit Klangerzeugung und Signalverarbeitung zu tun haben, sind oder sollten mit dem Audiosignal realisiert sein. In der Regel wird dies dem Anwender standardmäßig durch die verfügbaren Ports eines Moduls abgenommen. Bei der Verwendung eines Envelopes als Lautstärkehüllkurve muss der Nutzer ein entsprechendes Modul wählen, das einen Audioausgangsport besitzt. Audiosignale laufen in Reaktor mit der eingestellten Samplingrate und die entsprechenden Ports sind weiß eingefärbt.

2.1.2 Event

Eventsignale laufen mit einer `Control Rate` zwischen 25 und 3200 Hz. Der Standardwert beträgt 400 Hz. Die niedrigere Rate ermöglicht das Einsparen von Prozessorressourcen in Anwendungsfällen, in welchen man nicht auf eine hohe Auflösung angewiesen ist. Dazu gehören beispielsweise das Ändern der Tonhöhe, das Senden von Informationen an Displayelemente und das Übermitteln von Sequencerdaten. Eventports sind gelb eingefärbt.

2.1.3 Table Reference

Table Reference-Signale senden keine Daten sondern Referenzen auf zweidimensionale Arrays. Ein referenzierter Array kann in Core ausgelesen werden und dient als Grundlage für Sample basierte Algorithmen in Core und die Unterstützung von Ziehen und Ablegen in Primary. Table Reference-Ports sind durch ein rötliches Quadrat gekennzeichnet.

2.2 Fehlerdiagnose

Wie andere gängige integrierte Entwicklungsumgebungen, bietet Reaktor Mechanismen zum Diagnostizieren und Auffinden von Fehlern innerhalb der Programmstruktur. Mit aktiviertem `Wire Debugging` wird beim Überfahren einer Verbindung mit dem Mauszeiger der aktueller Wert der entsprechenden Strecke angezeigt. Dies eignet sich weitestgehend für die Analyse von Schalterzuständen oder andern nicht sequenziell getakteten Verbindungen.

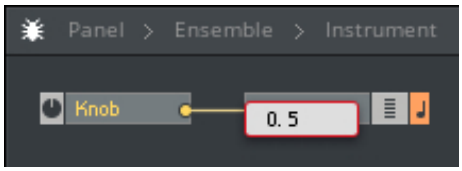

Abbildung 4: Wire Debugging

Bei aktivierter `Module Sorting` Anzeige wird dem Benutzer die Modulreihenfolge von Modulen mit aktiven Audioverbindungen angezeigt. Interessant ist dies besonders für Strukturen mit Rückführungspfad, in denen nicht direkt erkennbar ist, in welcher Abfolge die Signalverarbeitung erfolgt.

Des weiteren lässt sich die Reihenfolge anzeigen, in der Module mit Eventverbindungen bei Programmstart oder Laden eines Ensembles initialisiert werden. Der Signalfluss von Eventsignalen lässt sich über die Verwendung von Order-Modulen steuern, die darüber hinaus auch den Initialisierungsvorgang definieren.

2.2.1 Event Watcher, Audio Core Event Watcher

Für die Analyse von Werten über die Zeit, wird in der Factory Library der Event Watcher und in der User Library der Audio Core Event Watcher bereitgestellt. Beide Debugger basieren auf dem gleichen Prinzip die empfangenen Werte an ihren Eingangsports auf eine zeitliche Achse zu übertragen.

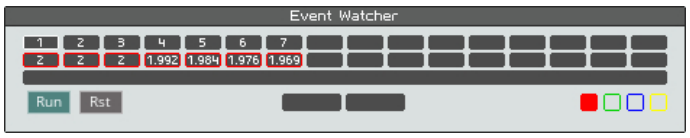

Abbildung 5: Factory Event Watcher

Der Audio Core Event Watcher für Primary besitzt eine detailliertere Benutzeroberfläche, die zusätzlich Initialisierungs- und Re-Initialisierungszeiträume markiert und Zeitpunkte und die dort empfangenen Events farblich gruppiert.

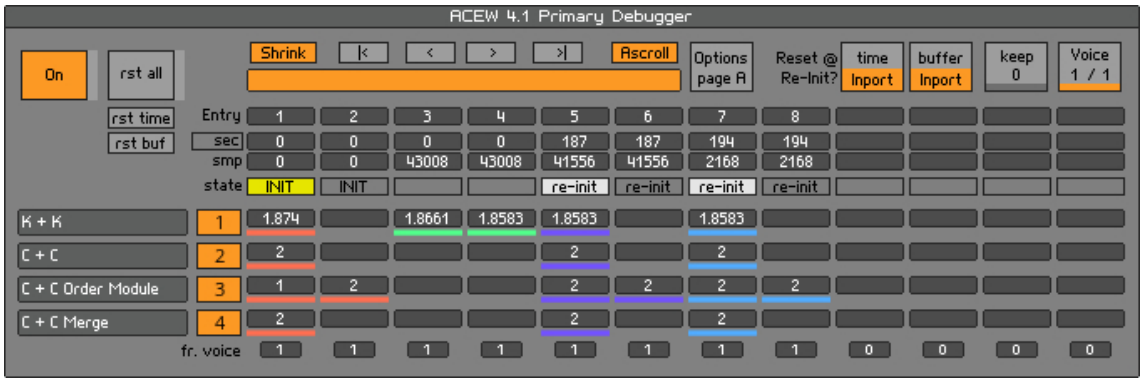

Abbildung 6: Audio Core Event Watcher

2.3 Init und Re-Init

Neben der Initialisierungsreihenfolge bei Programmstart und dem Aus-/Einschalten der Audioumgebung, kommt es bei Bearbeitung der dem Ensemble/Instrument zugrunde liegenden Struktur und dem Betätigen von `Switches` zu Re-Initialisierungsprozessen. Probleme können durch eine generelle Verwendung von `Order`-Modulen vermieden werden.

Ein besonderes (Re-)Initialisierungsverhalten zeigen das `Merge`- und das `Order`-Modul. Das Merge-Modul sendet bei `Init` und `Re-Init` nur den Wert des untersten Ports und das Order-Modul sendet zunächst den empfangenen Wert zum obersten Port und erst einen Schritt später zu den zwei anderen Ports.

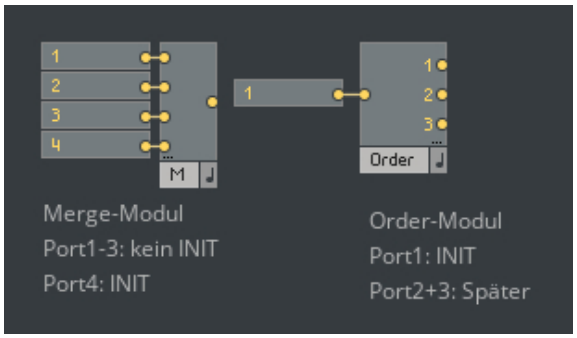

Abbildung 7: Merge- und Order-Modul

2.4 Send und Receive, Internal Connections

`Send` und `Receive`-Module dienen zum Senden und Empfangen von Werten über leitungslose Verbindungen. Die Werte eines sendenden Moduls, lassen sich bei beliebig vielen empfangenden Modulen über die Konfiguration im Moduleigenschaftsreiter und dem dort integrierten Funktionsreiter abgreifen.

Internal Connections ermöglichen das Verbinden von Panelelementen wie `Knobs` untereinander und/oder mit `IC-Send/Receive`-Modulen. Einstellungen werden im Verbindungsreiter des Moduleigenschaftsreiters vorgenommen. Des weiteren bieten IC-Sends die Möglichkeit durch eine Repräsentation auf dem Panel auf empfangende Elemente geroutet zu werden.

2.5 for-Schleife

Nachfolgend ist eine for-Schleife zum Berechnen einer Summe, die als Vergleich zur Programmiersprache Java dienen soll, abgebildet. Zentrales Elemente sind das `Iteration`-Modul, dass zum Erzeugen der Schritte und das `Accumulator`-Modul, dass zum Aufsummieren der Werte dient.

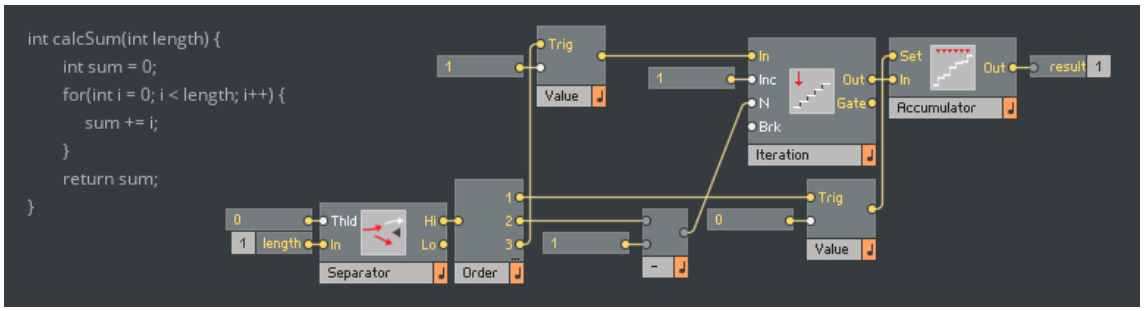

Abbildung 8: for-Schleife in Primary

2.6 Kontrollstruktur

Als weiterer Vergleich wird eine Kontrollstruktur herangezogen. Hier ist das zentrale Element das `Compare/Equal`-Modul, dass zwei Werte vergleichen und auf Gleichheit prüfen kann. Im abgebildeten Beispiel steuert das Ergebnis des größer Vergleichs die Portauswahl eines `Selectors`.

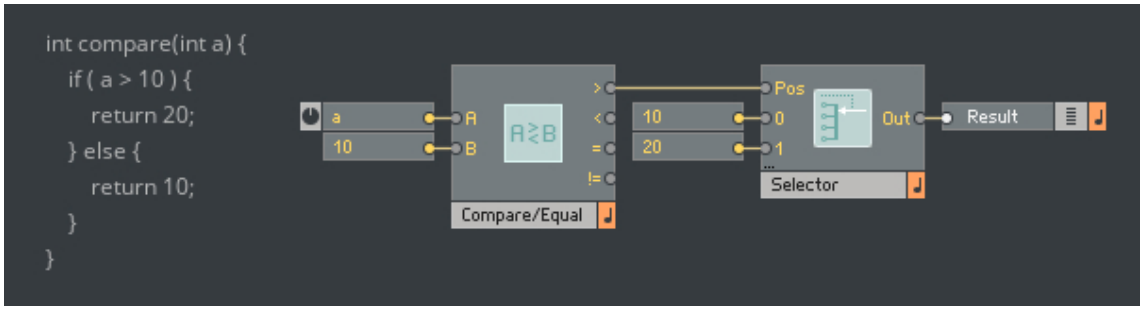

Abbildung 9: Kontrollstruktur in Primary

2.7 Arrays (Audio-/Event-Table)

Die Funktionalität von zweidimensionalen Arrays wird in der Primary-Ebene anhand von `Audio-` bzw. `Eventtables` umgesetzt. Ein Eventtable hat horizontal und vertikale Schreib- und Lesepositionen mit entsprechenden Schreib- und Leseports, die im einfachsten Fall per Iterationsmodul angesprochen werden. Werte innerhalb des Arrays können horizontal und vertikal linear interpoliert werden. Außerdem besitzen Audio- und Eventtables eine grafische Repräsentation auf der Panelebene und die Daten lassen sich exportieren und importieren.

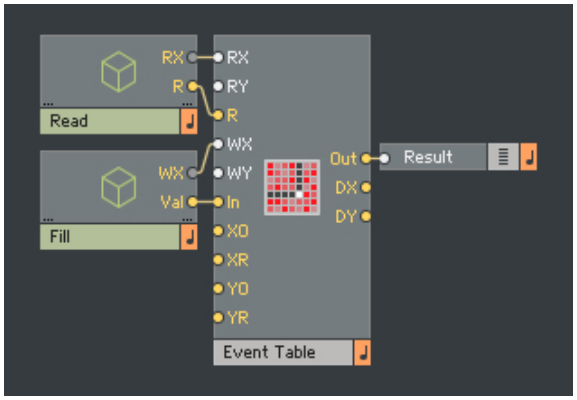

Abbildung 10: Eventtable in Primary

3 Die Core-Ebene

Im folgenden wird die Core-Ebene beschrieben. Zu ihr gelangt man über Core Cells die sich in der Primary-Ebene befinden. Sie ist damit die tieferliegende Entwicklungsebene, in der sich eigene Signalverarbeitungsalgorithmen und damit eigene Oszillatoren und Filter entwickeln lassen. In Core besteht keine Beziehung zur Panel-Ebene, wie in Primary kann jedoch anhand von vorgefertigten Makros oder dem `Filter`- und `Envelope-Toolkit` auf einer höheren Abstraktionsebene gearbeitet werden.

3.1 Signaltypen

In der Core-Ebene wird zwischen Scalaren, Object Bus Connections, Boolean Control, Bundles und Scoped Buses unterschieden. Auch in Core existieren für den entsprechenden Signaltyp eigene Verbindungen, die entsprechend unterschiedlich gekennzeichnet sind.

3.1.1 Skalare

Skalare sind innerhalb der Core-Ebene die am meisten verwendeten Signale. Sie können entweder ganzzahlig oder in Form von Gleitkommazahlen vorliegen und repräsentierten sowohl Audio- als auch Steuerdaten. In der Core-Ebene wird nicht zwischen Audio- und Eventsignalen unterschieden. Zwischen Floats und Integern findet eine automatische Konvertierung statt. Das Verhalten beim Runden von Zahlen wie 1.5 ist nicht definiert und muss explizit implementiert werden. Integerverbindungen werden mit bläulichen rechteckigen Ports, Floatverbindungen mit weißen runden Ports dargestellt.

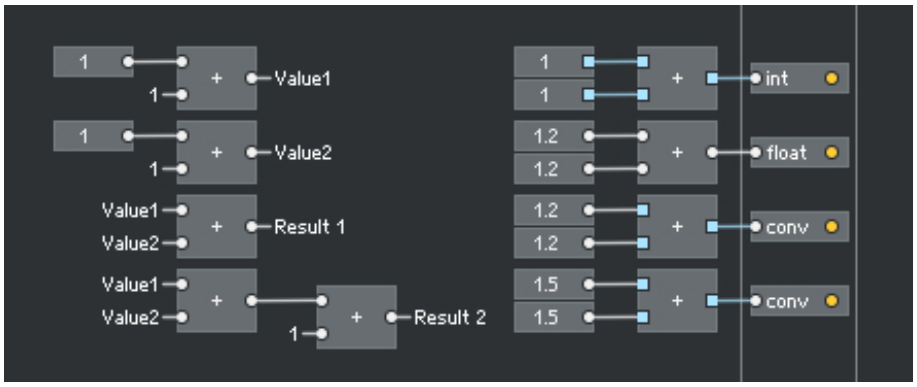

Abbildung 11: Skalare in Core

Linksseitig ist in der vorangegangenen Abbildung die Verwendung von `QuickConsts` und `QuickBusse` dargestellt. Sie verbessern die Lesbarkeit der Codestruktur und vereinfachen die Wiederverwendung bereits existierender Variablen, da die Verbindung zu einem QuickBus mit einem Kontextmenü gesetzt werden kann und keine manuelle Leitung gezogen werden muss.

3.1.2 Object Bus Connections

Object Bus Connections werden verwendet um Speicherbereiche zu identifizieren. Die OBC-Verbindungen sind ebenfalls entweder auf Ganzzahlen oder Gleitkommazahlen ausgelegt, im Gegensatz zu Skalaren aber untereinander nicht kompatibel. Mittels Lese- und Schreiboperationen können entweder Speicherbereiche für einzelne Variablen oder Arrays adressiert werden. Ganzzahlige OBC-Verbindungen werden ebenfalls eckig dargestellt, die der Gleitkommazahlen rund. Für Arrays wird das gedoppelte Portsymbol, für einzelne Variablen das einfache Portsymbol verwendet. In der nachstehenden Abbildung wird der Speicherbereich jeweils eines int und eines float Arrays allokiert, darauf folgen ein Index-, ein Schreib- und ein Lesemodul.

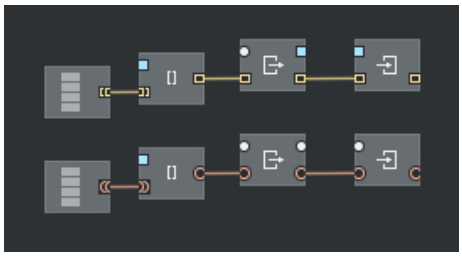

Abbildung 12: OBC in Core

3.1.3 Boolean Control

Boolean Control Signale werden innerhalb von Kontrollstrukturen als Steuersignale von Routern verwendet. Ports die BoolCtl senden und empfangen sind mit einem grünes Dreieck gekennzeichnet.

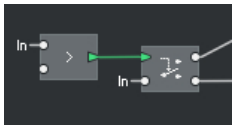

Abbildung 13:
BoolCtl in Core

3.1.4 Bundles

Bundles sind Containerverbindung von beliebigen Signalen und dienen zur Signalbündelung. Signale die innerhalb eines Bundles transportiert werden, werden über den vergebenen Namen der Ports identifiziert. Existiert innerhalb des Bundles kein Signal mit dem gewählten Portnamen oder existiert bereits ein Port mit demselben Namen, so wird der Port mit einem roten x versehen.

Abbildung 14: Bundles in Core

3.1.5 Scoped Buses

Scoped Buses sind Verbindungen, die vergleichbar mit den Send- und Receive-Modulen der Primary-Ebene sind. Sie dienen als leitungslose Verbindung von tieferen in höhere (Reception) und von höheren in tiefere Schichten (Distribution). Die Identifikation erfolgt hier ebenfalls wie bei Bundles über die Namensgebung.

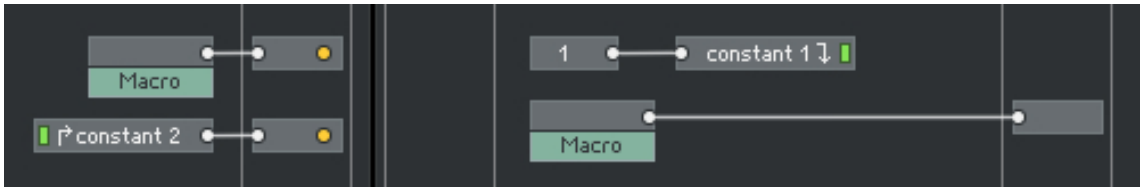

Abbildung 15: Scoped Busses in Core

3.2 Debugging

Das bereits im Kontext der Primary-Ebene genannte Wire Debbung und der erwähnte Audio Core Event Watcher können ebenfalls in Core verwendet werden bzw. im Fall des ACEWs ist dies der angedachte Anwendungsfall. Der ACEW ist die einzige Möglichkeit innerhalb der Core Cell Daten über die Zeit festzuhalten und auf der Panelebene zu analysieren. Der ACEW wird in der Primary-Ebene hinzugefügt und die Makros `CC_top_A`, `CC_btm_A` und `ACEW_A` werden entsprechend der Darstellung gesetzt und ein ACEW-Sensor, der die entsprechenden Signale entgegen nimmt, in die Core Cell eingefügt.

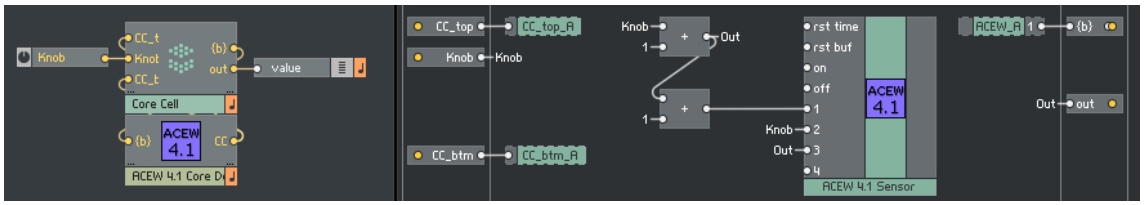

Abbildung 16: Audio Core Event Watcher in Core

3.3 Initialisierung

Core Cells werden in folgender Reihenfolge initialisiert: Zunächst wird der beschreibbare Speicher auf 0 zurückgesetzt (Object Bus Connections und Outputports), dann werden die Werte der Audio Inputs und anschließend die der Event Inputs abgegriffen, abschließend werden Konstanten, QuickConsts und Eingänge ohne Verbindungen initialisiert. In Core existiert im Gegensatz zu Primary kein Order-Modul, da die Reihenfolge des Datenflusses allein durch die Position innerhalb der Struktur definiert wird.

3.4 for-Schleife

In der Core-Ebene existiert aktuell kein Iteration-Modul und eine for-Schleife muss mit der Iteration Library aus dem Partials-Framework gelöst werden. Diese verwendet das Primary Iteration-Modul und bindet dessen Nutzung mit verschiedensten Arten an Iteratoren in die Core-Umgebung ein.

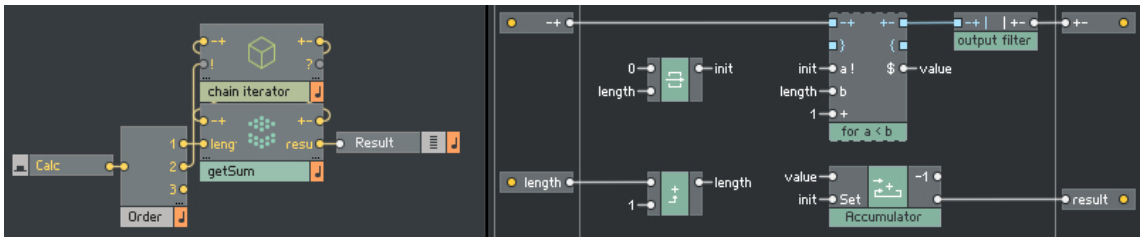

Abbildung 17: for-Schleife in Core

3.5 Kontrollstruktur

Die nachstehende Abbildung stellt die im Rahmen der Primary-Ebene gezeigte Kontrollstruktur übertragen in die Core-Ebene dar. Zentrale Bestandteile sind hier der `Router GT` (Greater Than Router) und die `Latches`, die bei ankommendem Signal ihren Wert weitergeben bzw. auf das ankommende Signal übertragen. Die genannten Makros sind in der Factory Library enthalten und können bei Bedarf geöffnet und angepasst werden.

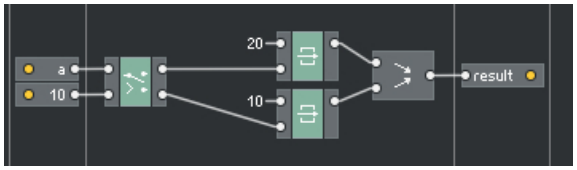

Abbildung 18: Kontrollstruktur in Core

3.6 Arrays und Tables

Nachstehend ist das Lesen von und Schreiben in einen Array abgebildet. Die per Object Bus Connection mit dem Array verbundenen Makros stellen sicher, dass die Auswahl des gesetzten Indexes und die Schreib- und Leseoperationen in der richtigen Reihenfolge ausgeführt werden. Neben den Arrays können alternativ auch Tables in die Core-Struktur eingefügt werden. Diese erlauben das Laden von Table - und Audio-Dateien.

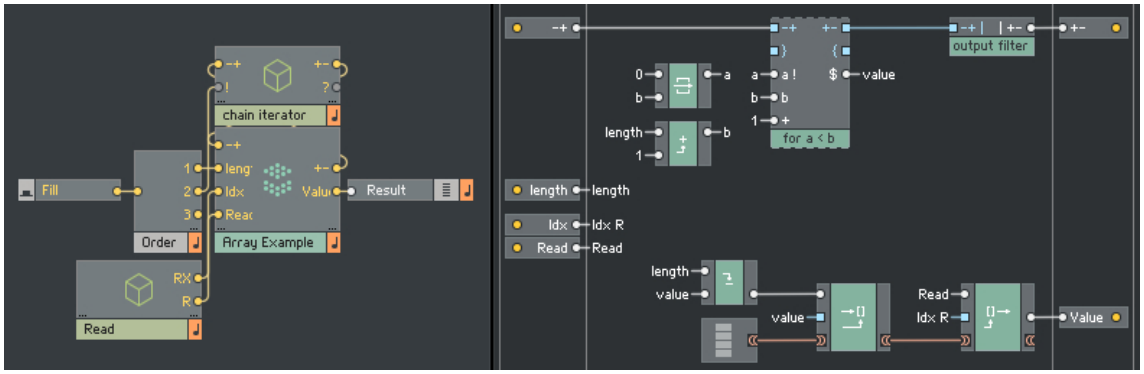

Abbildung 19: Array in Core

4 Fazit

Reaktor als Programmierumgebung und Werkzeug zum Erstellen eigener Instrumente und Effekte ist eine komplexe und vielschichtige Anwendung. Die Vielzahl an Modulen, deren Eigenheiten und der allgemeine Umgang mit der Reaktor Umgebung, Primary und Core gilt es zunächst zu erfassen und zu verstehen. Betrachtet man den Gesamtkontext, sind dann neben den genannten Grundkenntnissen je nach Umfang des eigenen Projekts auch Kenntnisse allgemeiner Algorithmen, Digitaler Signal Verarbeitung und User Interface Design nötig.

Attraktiv für Tonschaffende ist Reaktor vor allem auf Grund der Fülle und Vielfalt an Factory- und User Content. Es stehen rund 5000 fertige Ensembles zur Verfügung. Eine Brücke zwischen dem rein informatischen Aspekten und Reaktor als Musikinstrument und Sound Design Werkzeug wird dabei durch hochwertige bereits im Lieferumfang enthaltene DSP-Makros wie Oszillatoren, Filter und andere Effekte, sowie Steuerungsmodule wie Hüllkurven geschlagen. Ist ein Grundkenntnisstand erreicht, lassen sich damit bestehende Ensembles schneller an eigene Bedürfnisse anpassen. Der Grad der Komplexität lässt sich durch das vorhandene Material skalieren, da der Anwender zwischen der Verwendung von vorhandenem Material oder der Erstellung eigener Problemlösungen, wählen kann.

Literaturverzeichnis

Hanley, Adam (2017). *Reaktor 6: Building in Primary*. Native Instruments GmbH.

Hanley, Adam und Jan Ola Korte (2017). *Reaktor 6: Diving Deeper*. Native Instruments GmbH.

Hanley, Adam und Jan Ola Korte (2018). *Reaktor 6: Getting Started*. Native Instruments GmbH.

Zavalishin, Vadim (2015). *Reaktor 6: Buidling in Core*. Native Instruments GmbH.

Abbildungsverzeichnis

1 Benutzeroberfläche von Reaktor 6 1
2 Seitenleistenreiter zum Erstellen und Bearbeiten von Snapshots mit Zufalls- und Interpolationsfunktionen 3
3 Signaltypen der Primary-Ebene 4
4 Wire Debugging 5
5 Factory Event Watcher 6
6 Audio Core Event Watcher 6
7 Merge- und Order-Modul 7
8 for-Schleife in Primary 8
9 Kontrollstruktur in Primary 8
10 Eventtable in Primary 9
11 Skalare in Core 10
12 OBC in Core 11
13 BoolCtl in Core 11
14 Bundles in Core 12
15 Scoped Busses in Core 12
16 Audio Core Event Watcher in Core 13
17 for-Schleife in Core 13
18 Kontrollstruktur in Core 14
19 Array in Core 14