

Inhaltsverzeichnis

1 Architektur	7
1.1 Struktureller Aufbau	7
1.1.1 Systemstruktur	7
1.1.2 Architektur	8
1.2 Komponentenspezifikation	10
1.3 Schnittstellenspezifikation	12
2 Struktureller Softwareentwurf	16
2.1 Erläuterungen	16
2.1.1 Controller	17
2.1.2 Model	41
2.1.3 View	68
2.2 Klassendiagramme	94
2.2.1 View	94
2.2.2 Controller	99
2.2.3 Model	102
2.3 Patterns	107
3 Softwareentwurf für interaktive Elemente	113
3.1 Wichtige Sequenzdiagramme	113
4 Eigene Datenstrukturen	122
4.1 Befehlssatz im .json-Format	122
4.1.1 Mikroinstruktionsformat	122
4.1.2 Dateistruktur	123
5 Glossar	127
6 Anhang	130
6.1 Komplettes Klassendiagramm	130
6.2 Komplettes MIMA.json	130

Abbildungsverzeichnis

1.1	Systemstruktur	7
1.2	Mod-Struktur nach abgewandeltem Model-View-Controller-Architekturmuster	9
1.3	Komponentendiagramm	11
2.1	Paketdiagramm	16
2.2	1. Klassendiagramm View	94
2.3	2. Klassendiagramm View	95
2.4	Klassendiagramm Programming im View	96
2.5	Klassendiagramm Client	97
2.6	Klassendiagramm Items	98
2.7	Klassendiagramm Block-Controller	99
2.8	Klassendiagramm Simulation und Clustering	100
2.9	Klassendiagramm Assembler	101
2.10	Klassendiagramm Block-Model	102
2.11	Klassendiagramm Data	104
2.12	Klassendiagramm BusGraph und Memory	105
2.13	Klassendiagramm Instruction Set	106
2.14	Klassendiagramm Data Composite-Visitor-Pattern	107
2.15	Klassendiagramm der Model-View-Controller-Struktur für die einzelnen Blöcke am Beispiel des Register-Blocks	109
2.16	Klassendiagramm des kombinierten Command- und Visitor-Patterns für die Ausführung von MicroInstructions	110
2.17	Klassendiagramm des kombinierten Factory-Method- und Template-Method-Patterns für die Erstellung der Block-Models	111
2.18	Klassendiagramm des Observer-Patterns für die Aktualisierung des Zeitmodus der Simulation	112
3.1	<i>ComputerBlock <CL36></i> platzieren	113
3.2	Cluster auf Vollständigkeit prüfen und Simulation erstellen	114
3.3	<i>ComputerBlock <CL36></i> zerstört	115
3.4	<i>BusBlock</i> zerstört	116
3.5	Programm schreiben (View)	117
3.6	Programm schreiben (Controller)	118
3.7	Block Laden	119
3.8	Block Speichern	120
3.9	Ausführung eines Simulations-Ticks	121

4.1	Spezifikation der Basisinformationen des Befehlssatzes	123
4.2	Spezifikation der Register	124
4.3	Spezifikationen für Speicher-Größen	124
4.4	ALU-Operationen, die im Befehlssatz erlaubt sind	125
4.5	Ablauf der Holphase	125
4.6	Spezifikation von Befehlen	126
6.1	Klassendiagramm gesamt	130

1 Architektur

1.1 Struktureller Aufbau

1.1.1 Systemstruktur

Das System ist in zwei Dimensionen in Subsysteme aufgeteilt, wie in Abbildung 1.1 zu sehen. In der ersten Dimension liegt die für Minecraft typische Unterteilung in Server- und Client-Seite vor. Von der (logischen) Server-Seite wird die Welt angeboten und es werden Funktionen zur Verfügung gestellt und ausgeführt. Der (logische) Client auf der anderen Seite zeigt die Welt an und bietet so die Interaktionsschnittstelle für den Spieler. Die Kommunikation zwischen Client und Server funktioniert auf Basis von Anfragen. Dabei sendet der Client für eine bestimmte Aktion eine Anfrage an den Server. Dieser führt sie dann auf der Welt aus und informiert einen oder mehrere Clients über eventuell darzustellende Änderungen.

In der zweiten Dimension ist der native Minecraft-Teil der Software von der Modifikation (Mod) zu unterscheiden. Beide Teile liegen auf Client und Server vor, allerdings werden Softwareteile zur Bereitstellung der Programmlogik nur auf dem Server genutzt, während die Darstellung nur vom Client behandelt wird. Die Mod und Minecraft arbeiten eng zusammen, da alle Mod-Elemente von Minecraft darstellbar sein müssen und die von Minecraft angebotenen Interaktionsschnittstellen nutzen. Es ist zu beachten, dass die Interaktion der beiden Subsysteme dadurch sehr komplex und engmaschig ist, dies aber der vorgesehene Weg für Mods ist. Daher ist die Darstellung in Abbildung 1.1 stark vereinfacht.¹

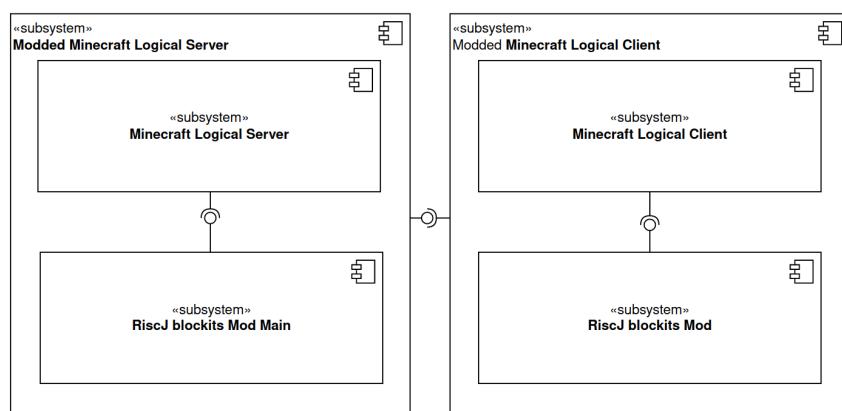


Abbildung 1.1: Systemstruktur

Abbildung 1.1 zeigt die Gesamtstruktur des Systems und damit insbesondere die Interaktion der Mod mit der Minecraft-Codebasis.

¹<https://fabricmc.net/wiki/tutorial:side>

1.1.2 Architektur

Die Mod arbeitet nach dem Model-View-Controller-Architekturmuster. Dieses bietet sich für das System in besonderer Weise an, da es sich um eine Modifikation mit visueller Nutzerschnittstelle, Steuerungsebene und einer komplexen darunterliegenden Datenverarbeitung handelt. Insbesondere lässt sich dies auch auf einen Computer, wie ihn die Mod simuliert, abbilden, da dieser eine Bedienoberfläche, eine Logik und einen hardwareseitigen Zustand besitzt. Das Model-View-Controller-Muster wird abgewandelt, um eine Anpassung an die Minecraft-Systemstruktur zu ermöglichen, wie in Abbildung 1.2 dargestellt ist. Die View-Komponente übernimmt dabei einige Aufgaben, die sonst vom Controller übernommen werden, wie z.B. Laden und Speichern. Diese Aufgaben wurden umverteilt, um Minecraft-native Operationen benutzen zu können. Insbesondere zu beachten ist dabei die architekturelle Aufteilung der View-Komponente. Diese liegt zwar sowohl im Client als auch im Server vor, allerdings sind die Subkomponenten der View-Komponente nicht vollständig in beiden Subsystemen aktiv. Die Client-Subkomponente des View findet sich nur im Client-Subsystem und übernimmt die Verarbeitung von Benutzeroberflächen. Die Main-Subkomponente des View findet sich in beiden Subsystemen und handhabt die Minecraft-Welt auf beiden Seiten. Dabei werden komplexere und Mod-spezifische Aktionen, welche sich dann auch auf Controller- und Model-Komponente auswirken, nur im Server-Subsystem ausgeführt.

PRAXIS DER SOFTWAREENTWICKLUNG
RISCJ Blockits

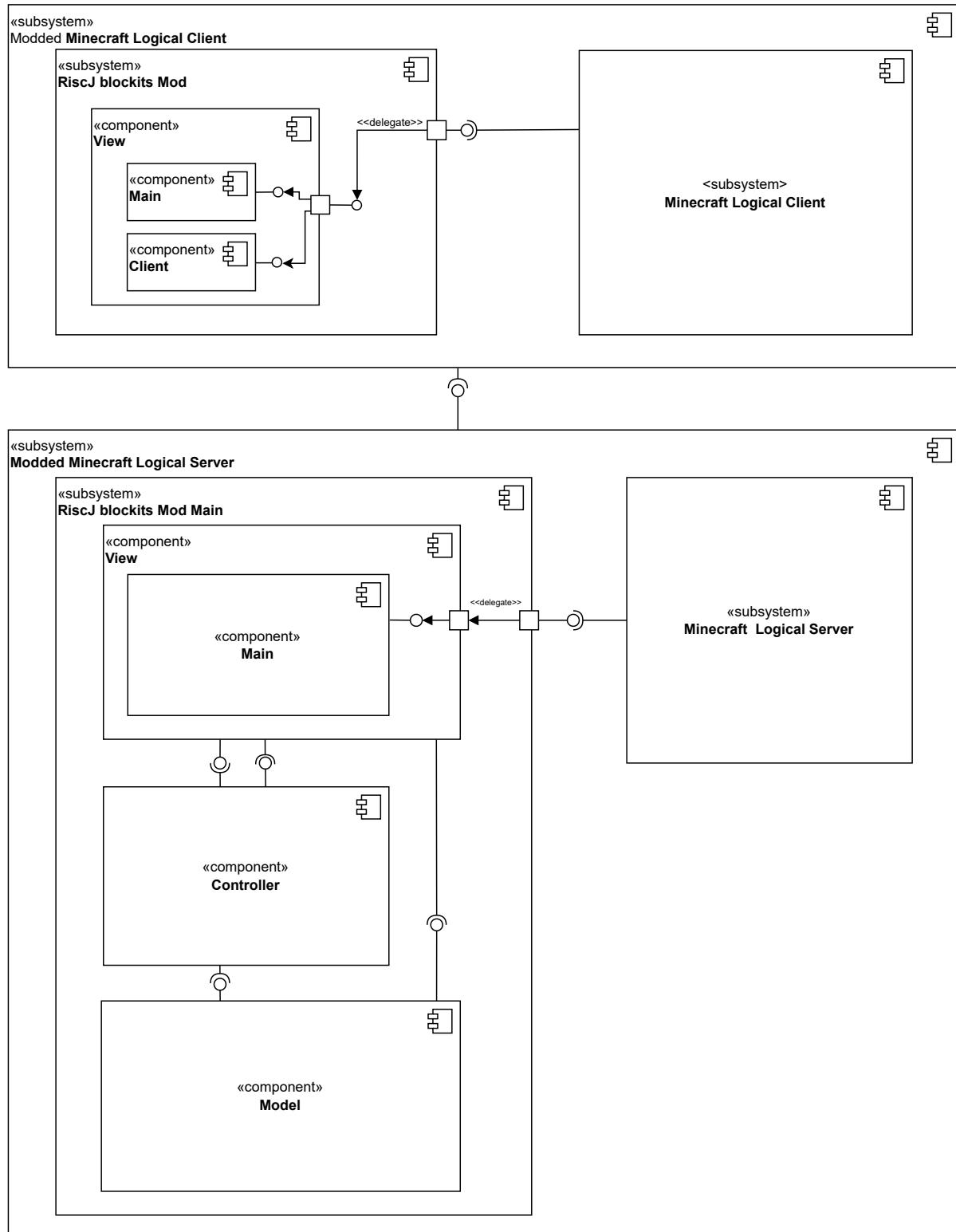


Abbildung 1.2: Mod-Struktur nach abgewandeltem Model-View-Controller-Architekturmuster

1.2 Komponentenspezifikation

Die Software der Mod ist in drei Hauptkomponenten eingeteilt. Diese sind in Abbildung 1.3 inklusive ihrer Subkomponenten dargestellt und werden im Folgenden erläutert.

View

Die View-Komponenten beinhalteten in der Main-Subkomponente die Darstellung der einzelnen Hardware-Komponenten des Computers in der Subsubkomponente *Computer Objects* aus Abbildung 1.3 sowie von sonstigen Computer-bezogenen Blöcken und Items, gesammelt in der Subsubkomponente *Utility Objects*. Die Client-Subkomponente des View beinhaltet in ihrer Subsubkomponente *Screen Elements* die Darstellung aller Graphischen-Nutzer-Oberflächen der Mod. Hervorzuheben ist insgesamt, dass die View-Komponente also die Visualisierung des Zustandes der Blöcke verwaltet, wodurch dem Spieler einen Einblick in die Vorgänge des gebauten Computers gewährt wird und somit das Ziel der Lehranwendung unterstützt.

Controller

Die Controller-Komponente beinhaltet die Block-Controller Komponente, die die Blöcke und die Models der Blöcke verwaltet und für die anderen Controller Komponenten Operationen bereitstellt. Hier findet sich auch die Logik für den Aufbau eines zusammenhängenden Lerncomputers, in der Abbildung 1.3 als *Clustering* benannt. Zudem findet sich dort die eigentliche *Simulation* des Computerverhaltens bei Programmausführung. Schließlich liegt im Controller auch die Logik für das Umwandeln eines Programmes in für den Lerncomputer verständlichen Maschinencode. Diese Subkomponente wird als *Assembler* bezeichnet.

Model

Die Model-Komponente beinhaltet die Daten, den Status und die Struktur des Lerncomputers. Die räumliche Struktur wird wie in Abbildung 1.3 in der Komponente *BusGraph* gespeichert. Dadurch weiß der Computer, welche Teile zu ihm gehören und wo sie liegen. Die *Instruction Set*-Komponente legt fest, wie sich der Computer bei Eingaben verhält. Sie entscheidet, wie der Computer Programme ausführt und übersetzt. Die Speicherung des Zustands aller Komponenten findet in der *Hardware Elements*-Komponente statt. Jeder Minecraft-Block der Mod hat hier seine Repräsentation.

PRAXIS DER SOFTWAREENTWICKLUNG

RISCJ Blockits

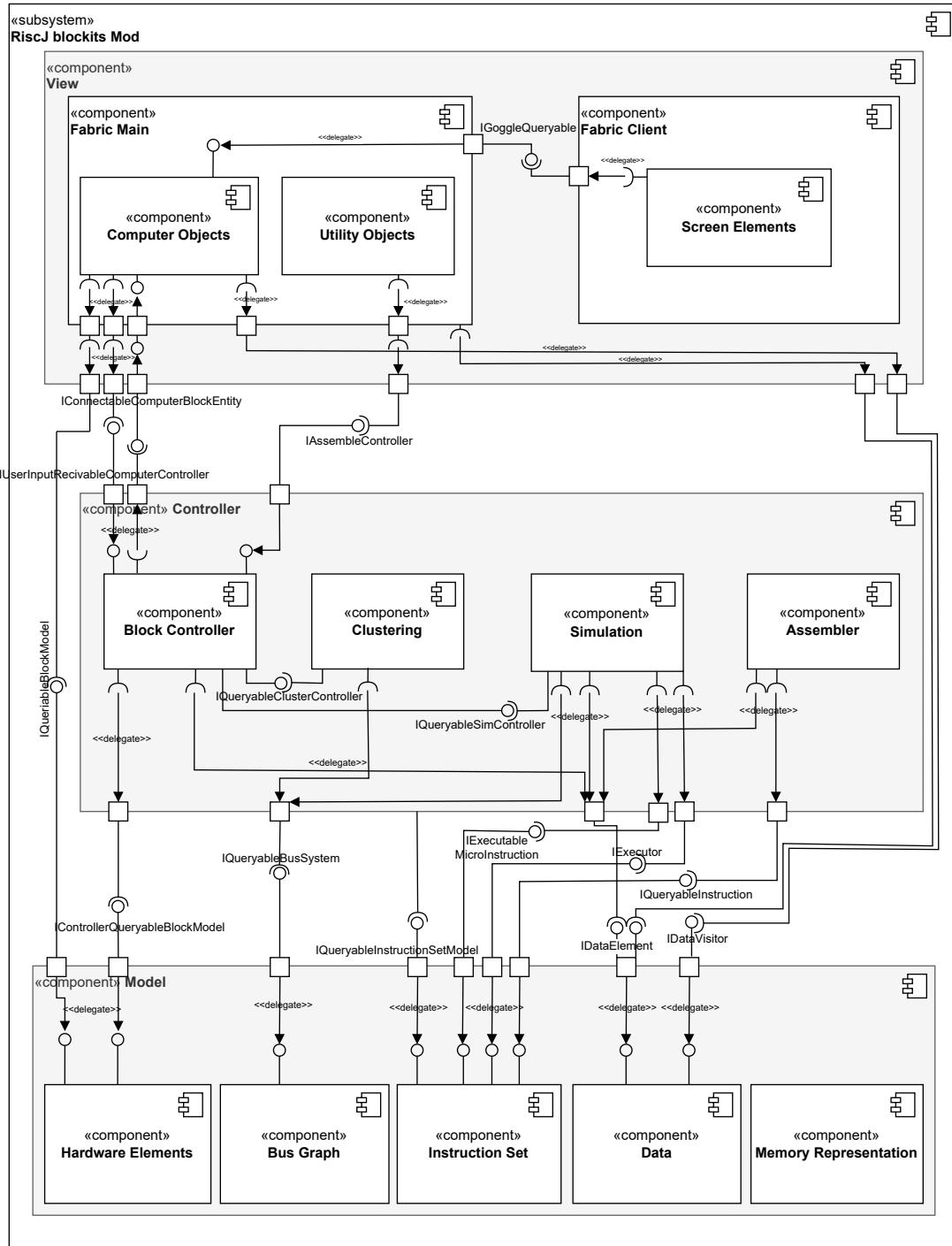


Abbildung 1.3: Komponentendiagramm

1.3 Schnittstellenspezifikation

IConnectableComputerBlockEntity

Provided von: Computer-Objects-Komponente, genauer alle *ComputerBlockEntitys*.

Required von: *BlockController* Komponente, genauer alle *ComputerBlockController*.

Stellt den Zugriffspunkt für *BlockController* auf ihre *BlockEntitys* bereit.

IUserInputReceivableComputerController

Provided von: Block-Controller-Komponente, genauer alle *ComputerBlockController*.

Required von: Computer-Objects-Komponente, genauer alle *ComputerBlockEntitys*.

Bietet Funktionalität für die Übergabe von Benutzereingaben, beispielsweise Minecraft-Ticks, von *BlockEntitys* an ihre Controller.

IAssemblerController

Provided von: Block Controller Komponente, genauer der *ProgrammingBlockController*.

Required von: Utility Objects Komponente, genauer die *ProgrammingBlockEntity*.

Bietet Funktionalität für die Kommunikation zwischen dem *ProgrammingBlock* und seinem Controller, wie z.B. vom Benutzer eingegebene Programme zusammenzubauen.

IQueryableBlockModel

Provided von: Hardware-Elements-Komponente

Required von: Computer-Objects-Komponente

Stellt den Zugriffspunkt für *ComputerBlockEntitys* bereit, um ihren Status beim Model anzufragen, welcher dann dem Benutzer dargestellt werden kann.

IGoggleQueryable

Provided von: Computer-Objects, genauer die *ComputerBlockEntitys*.

Required von: Screen-Elements, genauer der *GoggleUI*.

Stellt Funktionalität für die Brille bereit, mit der sie sich Informationen zum Anzeigen holen kann.

IQueryableSimController

Provided von: Block-Controller-Komponente, genauer die *ComputerBlockController*.

Required von: Simulation-Komponente.

Stellt Informationen und allgemeine Operationen auf ComputerBlockControllern bereit.

IQueryableClusterController

Provided von: Block-Controller-Komponente, genauer die *ComputerBlockController*.

Required von: Clustering-Komponente

Bietet der Clustering-Komponente Zugriff auf die *ComputerBlockController*, damit Blöcke richtig in der Graphenrepräsentation des Computers eingeordnet werden kann.

IControllerQueryableBlockModel

Provided von: Hardware-Elements-Komponente, genauer alle *BlockModels*.

Required von: Block-Controller-Komponente, genauer alle *ComputerBlockController*.

Dient zur Festlegung der allgemeinen Kommunikation von Controllern mit ihren Modellen.

IQueryableBusSystem

Provided von: Bus-Graph-Komponente, genauer das *BusSystemModel*.

Required von: Clustering-Komponente, genauer dem *ClusterHandler*

Definiert, wie das Clustering Operationen auf dem Model des Computeraufbaus durchführen kann. Das Model wird als Graph gespeichert.

IExecutableMicroInstruction

Provided von: Instruction Set Komponente, genauer alle *MicroInstructions*.

Required von: Simulation-Komponente

Definiert, wie die Simulation einzelne Simulationsschritte ausführt. Eine unserer *Micro-Instruktionen* bildet einen Mima Takt.

IQueryableInstruction

Provided von: Instruction-Set-Komponente, genauer der Instruction.

Required von: Assembler-Komponente

Definiert den Umgang beim assembeln mit Instruktionen. Eine Instruction repräsentiert einen Assembler Befehl.

IQueryableInstructionSetModel

Provided von: Instruction-Set, genauer das *InstructionSetModel*

Required von: Controller-Komponente

Stellt Funktionalität bereit, mit der der Computer abgleichen kann, ob er auch mit der definierten Architektur übereinstimmt, welche Operationen er dann bereitstellen soll und wie sie definiert sind.

IExecutor

Provided von: Instruction-Set-Komponente

Required von: Simulation-Komponente

Stellt Funktionalität bereit, um *MicroInstructions* zu besuchen und auszuführen.

Teil des Visitor-Patterns für die Programmausführung. Aus der Simulation-Komponente werden verschiedene *MicroInstruktion* aus der *InstructionSet* Komponente besucht, um sie auszuführen.

ISimulationTimingObservable

Provided von: *SystemClockModel*

Required von: *SimulationTimeHandler*

Stellt Funktionalität bereit, um Observer zu verwalten und zu benachrichtigen.

Teil des Observer-Patterns, mit dem die Simulations-Komponente jeden Takt vom *SystemClockController* mitgeteilt bekommt, dass ein Minecraft Takt vergangen ist. Das muss variabel passieren, da ein Minecraft-Takt je nach Systemlast nicht immer genau gleich lang ist.

ISimulationTimingObserver

Provided von: *SimulationTimeHandler*

Required von: *SystemClockModel*

Stellt Funktionalität bereit, um Nachrichten von beobachteten Elementen entgegenzunehmen.

Teil des Observer-Patterns, mit dem die Simulations-Komponente jeden Takt vom *SystemClockController* mitgeteilt bekommt, dass ein Minecraft Takt vergangen ist. Das muss variabel passieren, da ein Minecraft-Takt je nach Systemlast nicht immer genau gleich lang ist.

IDataElement, IDataContainer, IDataEntry, IDataStringEntry

Provided von: Data-Komponente

Required von: View-Komponente und Controller-Komponente

Stellt Funktionalität bereit, um Daten universell auszutauschen.

Teil des Composite-Patterns, das als universelles Daten-Format benutzt wird. Container und Entry sind Elemente, StringEntry ein Entry

IDataVisitor

Provided von: Data-Komponente

Required von: Fabric-Main-Komponente

Dient der Umwandlung von *Data* zu *NbtElement*, mit welchem Minecraft arbeitet. Teil des Visitor-Patterns, das es erlaubt, *IDataElemente* zu besuchen.

2 Struktureller Softwareentwurf

In das Komponentendiagramm der Mod lassen sich die Pakete, in die die Software gegliedert ist, siehe Abbildung 2.1, einbetten.

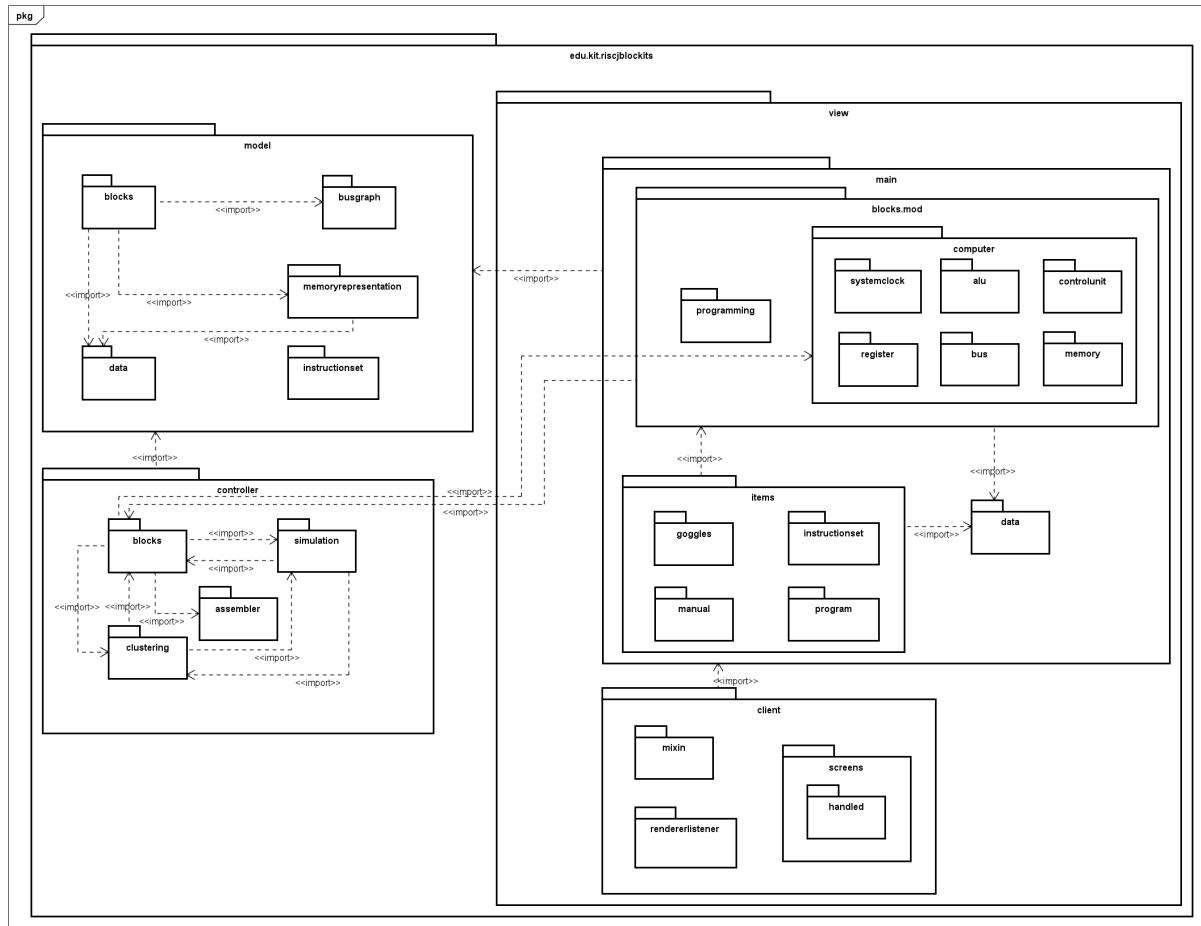


Abbildung 2.1: Paketdiagramm

2.1 Erläuterungen

Nicht alle Klassen und Methoden werden beschrieben, sondern nur eine Auswahl der wichtigen. Eine komplette Aufzählung ist im Klassendiagramm zu finden. Die Klassenbeschreibungen wurden teilweise aus Javadoc¹ generiert und mit DeepL übersetzt. Diese Skelette wurden dann händisch überarbeitet und umformuliert.

¹teilweise mit Unterstützung von Github-Copilot

2.1.1 Controller

Assembler

Assembler $\langle CL1 \rangle$

Aufgabe

Diese Klasse stellt einen Assembler dar, der Assemblercode in Maschinencode übersetzt. Er schreibt den Maschinencode in einen *Memory*.

Attribute

Die Klasse stellt folgende Attribute mit Sichtbarkeit *private* bereit:

Name	Typ	Einschränkungen	Beschreibung
LABEL_-COMMAND_-PATTERN	Pattern		Regex-Muster zur Trennung von Label und Befehl.
instructionSet-Model	IQuerable-InstructionSet-Model		Das für das Assembeln zu verwendende <i>InstructionSetModel</i> . Wird dem Konstruktor als Argument übergeben.
memory	Memory		Der Speicher, in die der assemblte Code geschrieben wird.
current-Address	Value		Die aktuelle Adresse, an die geschrieben wird.
calculated-Memory-AddressSize	int		Die Größe der Speicheradresse in Bytes.
calculated-MemoryWord-Size	int		Die Größe der Speicherwörter in Bytes.
labels	Map<String, Value>		Die Map die Beschriftungen Adressen zuweist.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
assemble	void	assemblyCode: String	Assembelt den angegebenen Assembler-Code und schreibt ihn in ein <i>Memory</i> . Der Parameter <i>AssemblyCode</i> enthält den Assemblercode, der assembelt werden soll. Die Methode wirft eine <i>AssemblyException</i> , wenn der Assembler-Code nicht assembelt werden kann.
getMemory- Data	IDataElement		Gibt die Daten des <i>Memory</i> , in den geschrieben wurde, wenn der Code assembelt wurde. Der Rückgabewert gibt den <i>Memory</i> zurück, in den geschrieben wurde.

Die Klasse stellt folgende Methoden mit Sichtbarkeit *private* bereit²:

Name	Rückgabetyp	Parameter	Beschreibung
getCommand- ForLine	Command	line: String	Ruft den <i>Command</i> für eine bestimmte Zeile ab. Erkennt und ersetzt auch Labels. Der Parameter <i>line</i> enthält die Zeile, für die der Befehl abgerufen werden soll. Der Rückgabewert gibt den Befehl für die angegebene Zeile zurück. Wirft eine <i>AssemblyException</i> , wenn der Befehl nicht assembelt werden kann.

²ggfs. werden private Hilfsmethoden nicht dargestellt

Name	Rückgabetyp	Parameter	Beschreibung
extractLabel- Positions	void	lines: String[]	Extrahiert die Adressen aller Labels im Assemblercode. Dies geschieht durch Iteration über alle Befehle, unter Berücksichtigung von Adressänderungen. Der Parameter <i>lines</i> enthält die Assemblerzeilen, die auf Labels geprüft werden sollen.
writeLabels- ToArguments	void	arguments: String[]	Verwendet die erkannten Labels, um diese einzusetzen, wenn sie als Argument angegeben werden. Der Parameter <i>arguments</i> enthält eine Reihe von Argumenten, die zu ersetzende Bezeichnungen haben könnten.
write- RegistersTo- Arguments	void	arguments: String[]	Schreibt die Registeradressen des Befehlssatzes in die Argumente. Der Parameter <i>arguments</i> enthält eine Reihe von Argumenten, die Register haben könnten, die ersetzt werden müssen.

Zugehörigkeit

Die Klasse befindet sich im Paket *controller.assembler* und nutzt die Klassen *model.memoryrepresentation.Memory*, *model.memoryrepresentation.Value*, *model.data.IDataElement*, *model.instructionset.IQueryableInstruction*, *model.instructionset.IQueryableInstructionSetModel*.

Command $\langle CL2 \rangle$ **Aufgabe**

Diese Klasse stellt einen einzelnen Befehl im Assemblercode dar.

Attribute

Die Klasse stellt folgende Attribute mit Sichtbarkeit *private* bereit:

Name	Typ	Einschränkungen	Beschreibung
ARGUMENT_ TRANS- LATION_ PATTERN	Pattern		Regex-Muster, um Teile der Argumentübersetzung zu extrahieren.
assembled- Translation	String[]		Die zusammengesetzte Übersetzung des Befehls.
arguments- Instruction- Map	Map<String, String>		Zuordnung von Argumenten zu ihren Werten. Wird im Konstruktor über ein Argument erstellt.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
asValue	Value		Gibt die zusammengesetzte Übersetzung des Befehls als <i>Value</i> zurück.

Die Klasse stellt folgende Methoden mit Sichtbarkeit *private* bereit³:

Name	Rückgabetyp	Parameter	Beschreibung
assemble-Argument	String	translationPart: String	Setzt einen einzelnen Übersetzungsteil zusammen. Hierfür wird der Wert in binär umgewandelt und auf die benötige Länge gekürzt. Der Parameter <i>translationPart</i> enthält den Übersetzungsteil, der zusammengesetzt wird. Der Rückgabewert gibt den zusammengesetzten Übersetzungsteil zurück. Wirft eine <i>AssemblyException</i> , wenn der Übersetzungsteil nicht zusammengesetzt werden kann.

Zugehörigkeit

Die Klasse befindet sich im Paket *controller.assembler* und nutzt die Klassen *model.memoryrepresentation.Value*, *model.instructionset.IQueryableInstruction*.

³ggfs. werden private Hilfsmethoden nicht dargestellt

ValueExtractor $\langle CL3 \rangle$

Aufgabe

Diese Hilfs-Klasse ermöglicht das Umwandeln verschiedener Zahlenwerte (als String) in *Value*-Objekte.

Attribute

Die Klasse stellt folgende Attribute mit Sichtbarkeit *private* bereit:

Name	Typ	Einschränkungen	Beschreibung
HEX_- VALUE_- PATTERN	Pattern		Regex-Muster für die Übereinstimmung mit Hex-Werten.
DEC_- VALUE_- PATTERN	Pattern		Regex-Muster für die Übereinstimmung mit dezimalen Werten.
BIN_- VALUE_- PATTERN	Pattern		Regex-Muster für die Übereinstimmung mit Binärwerten.
FLOAT_- VALUE_- PATTERN	Pattern		Regex-Muster für die Übereinstimmung mit Fließkommazahlen.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
extractValue	Value	value: String, length: int	static Extrahiert einen Wert aus einer Zeichenkette. Kann Hexadezimale (0x), binäre (0b), dezimale und Fließkomma Werte extrahieren. Der Parameter <i>value</i> enthält die Zeichenfolge, aus der der Wert extrahiert werden soll. Der Parameter <i>length</i> gibt die Länge des Wertes in Bytes an. Der Rückgabewert gibt den extrahierten Wert zurück oder "null", wenn der Wert nicht extrahiert werden kann.

Zugehörigkeit

Die Klasse befindet sich im Paket *controller.assembler* und nutzt die Klassen *model.memoryrepresentation.Value*.

Clustering und BlockController

BlockController<CL4>

Aufgabe

Definiert alle Controller. Jede *ModBlockEntity* hat einen Controller. Implementiert *IUserInputReceivableController*.

Attribute

Die Klasse stellt folgende Attribute mit Sichtbarkeit *private* bereit:

Name	Typ	Einschränkungen	Beschreibung
controllerType	BlockController-Type		Typ des Controllers. Er wird zur Unterscheidung in einer Liste von Controllern verwendet.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
setData	void	data: IData-Element	Wird von der Ansicht verwendet, wenn sie Daten im Modell aktualisieren will. Ausgelöst durch ein Neuladen des Minecraft-Blocks oder eine Benutzereingabe.
getController-Type	BlockController-Type		Getter für den Controller-Typ.

Die Klasse stellt folgende Methoden mit Sichtbarkeit *protected* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
setController-Type	void	controllerType: BlockController-Type	Der Parameter <i>controllerType</i> legt einen neuen Controllertyp fest.

Zugehörigkeit

Die Klasse befindet sich im Paket *controller.blocks* und nutzt die Klassen *model.data.IDataElement*.

Erbende Klassen

Von der Klasse erben der *ComputerBlockController* und der *ProgrammingController*. Der *ProgrammingController* verbindet sich nicht mit einem Computer, da er nicht Teil der Simulation ist. Der *ProgrammingController* stellt Funktionalität zum Compilieren von Assembler-Programmen bereit.

ComputerBlockController<CL5>

Aufgabe

Definiert alle *BlockController*, die an der Simulation teilnehmen. Jedes *ComputerblockEntity* hat einen *ComputerBlockController*. Implementiert *IUserInputReceivableComputerController*, *IQueryableClusterController* und *IQueryableSimController*. Erbt vom *BlockController*.

Attribute

Die Klasse stellt folgende Attribute mit Sichtbarkeit *private* bereit:

Name	Typ	Einschränkungen	Beschreibung
blockModel	IController- QueryableBlock- Model		Das Modell des Blocks. Jeder Controller der Teil der Simulation ist, speichert ein Modell, um die Daten des Blocks zu speichern.
blockEntity	IConnectable- ComputerBlock- Entity		Die <i>BlockEntity</i> , für die der Controller zuständig ist.
clusterHandler	ClusterHandler		Der <i>ClusterHandler</i> , der für den Cluster zuständig ist, in dem sich der Block befindet. Ein Cluster stellt mehrere Computerblöcke dar, die durch Busblöcke verbunden sind. Sie bilden einen Computer.
pos	BlockPosition		Die Position des Blocks in der Minecraft-Welt.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
setData	void	data: IData-Element	Wird von der Ansicht verwendet, wenn sie Daten im Model aktualisieren will. Der Parameter <i>data</i> enthält die Daten, die gesetzt werden sollen.
getModel	IController-QueryableViewBlock-Model		Wird als Controller verwendet, um das Model anderer Controller zu erhalten.
getNeighbours	List<Computer-BlockController->		Sammelt die Nachbarn des Blocks. Gibt nur <i>BusBlocks</i> zurück, wenn der Block ein <i>ComputeBlock</i> ist. Gibt alle <i>ComputeBlocks</i> zurück, wenn der Block ein <i>BusBlock</i> ist. Gibt eine leere Liste zurück, wenn der Block kein <i>ComputerBlock</i> ist.
setCluster-Handler	void	clusterHandler: ClusterHandler	Setzt den zugehörigen <i>ClusterHandler</i> . Der Parameter <i>clusterHandler</i> enthält den <i>ClusterHandler</i> , der mit dem Controller verknüpft werden soll.
onBroken	void		Wird von der <i>BlockEntity</i> aufgerufen, während der Block zerstört wird. Entfernt den Block aus dem Cluster.
tick	void		Wird bei jedem Tick von der <i>BlockEntity</i> aufgerufen. Überschrieben durch den <i>SystemClockController</i> .

Die Klasse stellt folgende Methoden mit Sichtbarkeit *protected* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
Computer-Block-Controller	ComputerBlock-Controller	blockEntity: IConnectable-ComputerBlock-Entity	Erzeugt <i>ClusterHandler</i> und Model.
createBlock-Model	IController-QueryableViewBlock-Model		abstract Erzeugt das Modell für den Block. Jeder Blocktyp hat seine eigene Art von Modell. Der Rückgabewert gibt ein blockspezifisches Modell zurück.

Zugehörigkeit

Die Klasse befindet sich im Paket *controller.blocks* und nutzt die Klassen *controller.clustering.-ClusterHandler*, *model.blocks.IControllerQueryableViewBlockModel*, *model.data.IDataElement*, *model.blocks.BlockPosition*, *model.blocks.IQueryableViewBlockModel*.

Erbende Klassen

- *AluController* bietet zusätzlich die Methode mit Sichtbarkeit *public* an:

Name	Rückgabetyp	Parameter	Beschreibung
execute-AluOperation	void	operation: String	Führt eine Operation auf dem <i>AluModel</i> aus. <i>operation</i> muss einer der unterstützten Operationen sein. Wirft eine <i>InvalidAluOperationException</i> falls der String nicht unterstützt wird.

- *BusController*

- *ControllUnitController* bietet zusätzlich die Methode mit Sichtbarkeit *public* an:

Name	Rückgabetyp	Parameter	Beschreibung
getInstruction-SetModel	Instruction-SetModel		Getter für den Befehlssatz.

- *MemoryController* bietet zusätzlich die Methode mit Sichtbarkeit *public* an:

Name	Rückgabetyp	Parameter	Beschreibung
getValue	Value	address: Value	Getter für einen Wert.
writeValue	void	address: Value, value: Value	Setter für den Befehlssatz.

- *RegisterController* bietet zusätzlich die Methode mit Sichtbarkeit *public* an:

Name	Rückgabetyp	Parameter	Beschreibung
getRegister-Type	String		Getter für den Typ. Die Typen sind im Befehlssatz definiert.
getValue	Value		Getter für den Wert im Register.
setNewValue	void	value: Value	Setter für den aktuellen Wert.

- *SystemClockController* bietet zusätzlich die Methode mit Sichtbarkeit *public* an:

Name	Rückgabetyp	Parameter	Beschreibung
tick	void		Gibt das Ticksignal an den <i>SimulationTimeHandler</i> weiter
setSimulation-TimeHandler	void	simulation-TimeHandler: Simulation-TimeHandler	Setzt den <i>SimulationTimeHandler</i> an den Ticks weitergegeben werden sollen.

ClusterHandler $\langle CL6 \rangle$

Aufgabe

Verwaltet eine Menge von *BlockControllern*, die zusammen ein Cluster bilden. Zudem verwaltet es den Graphen mit den *BlockPositionen* im *BusSystemModel*. Falls eine Architektur valide ist, baut der *ClusterHandler* die Simulation auf. Implementiert *IArchitectureCheckable*.

Attribute

Die Klasse stellt folgende Attribute mit Sichtbarkeit *private* bereit:

Name	Typ	Einschränkungen	Beschreibung
blocks	List<Computer-BlockController->		Liste aller <i>BlockController</i> , außer den <i>BusBlockControllern</i> , in diesem Cluster.
busBlocks	List<Computer-BlockController->		Liste aller <i>BusBlockController</i> in diesem Cluster.
busSystem-Model	BusSystem-Model		Das <i>BusSystemModel</i> des Clusters.
istModel	IQuerable-InstructionSet-Model		Das <i>InstructionSetModel</i> des Clusters.
building-Finished	boolean		True, wenn der Aufbau des Clusters abgeschlossen ist.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
combine	void	ownBlock: Computer- BlockController, neighbourBlock: ComputerBlock- Controller, old- Cluster: Cluster- Handler	Kombiniert dieses Cluster mit einem anderen Cluster. Dabei werden alle <i>BlockController</i> vom <i>oldCluster</i> übernommen. Außerdem werden die <i>BusSystemModels</i> der beiden Cluster kombiniert und die Graphen der <i>BusSystemModels</i> zwischen den Knoten von <i>ownBlock</i> und <i>neighbourBlock</i> mit einer Kante verbunden. Der Parameter <i>ownBlock</i> enthält einen <i>BlockController</i> dieses Clusters. Der Parameter <i>neighbourBlock</i> enthält einen <i>BlockController</i> von <i>oldCluster</i> .
block- Destroyed	void	destroyedBlock- Controller: ComputerBlock- Controller	Verwaltet die Zerstörung eines Blocks und die entsprechende Änderung im Cluster und im <i>BusSystemModel</i> . Falls der zerstörte Block das Cluster trennt, werden neue <i>ClusterHandler</i> erstellt. Der Parameter <i>destroyedBlockController</i> enthält den <i>BlockController</i> des zerstörten Blocks.

Name	Rückgabetyp	Parameter	Beschreibung
addBlocks	void	blockController: ComputerBlock- Controller	Methode, um dem Cluster einen <i>BlockController</i> hinzuzufügen. Der Parameter <i>blockController</i> enthält den hinzuzufügenden <i>BlockController</i> .
addBusBlocks	void	blockController: ComputerBlock- Controller	Methode, um dem Cluster einen <i>BusBlockController</i> hinzuzufügen. Der Parameter <i>blockController</i> enthält den hinzuzufügenden <i>BusBlockController</i> .
getBusSystem- Model	BusSystem- Model		Methode, um das <i>BusSystemModel</i> des Clusters zu erhalten. Der Rückgabewert ist das <i>BusSystemModel</i> des Clusters.
getBlocks	List<Computer- BlockController- >		Methode, um alle <i>BlockController</i> , außer den <i>BusBlockControllern</i> des Clusters zu erhalten. Der Rückgabewert gibt das Attribut <i>blocks</i> zurück.
getBusBlocks	List<Computer- BlockController- >		Methode, um alle <i>BusBlockController</i> des Clusters zu erhalten. Der Rückgabewert gibt das Attribut <i>busBlocks</i> zurück.
checkFinished	void		Methode, um zu prüfen, ob der Aufbau des Clusters abgeschlossen ist. Falls der Aufbau valide ist wird die Simulation gestartet.
start- Simulation	void		Startet die Simulation mit dem aktuellen Cluster.

Die Klasse stellt folgende Methoden mit Sichtbarkeit *private* bereit⁴:

Name	Rückgabetyp	Parameter	Beschreibung
combineTo- Neighbours	void	blockController: ComputerBlock- Controller	Kombiniert dieses Cluster mit allen NachbarClustern vom <i>blockController</i> . Wirft eine <i>ClusterToLargeException</i> falls das Cluster bei dem kombinieren größer als 200 Blöcke wird.

Zugehörigkeit

Die Klasse befindet sich im Paket *controller.clustering* und nutzt die Klassen *controller.blocks.-BlockControllerType*, *controller.blocks.IQueryableClusterController*, *controller.blocks.IQueryableComputerController*, *controller.blocks.IQueryableSimController*, *controller.blocks.SystemClockController*, *controller.simulation.SimulationTimeHandler*, *model.busgraph.BusSystemModel*, *model.busgraph.IQueryableBusSystem*, *model.instructionset.-IQueryableInstructionSetModel*, *model.instructionset.InstructionSetBuilder*.

Simulation

Executor⟨CL7⟩

Aufgabe

Steuert die Ausführung von Mikroinstruktionen. Wird vom *SimulationSequenceHandler* aufgerufen, wenn eine Mikroinstruktion ausgeführt werden soll. Unterscheidet zwischen den verschiedenen Typen von Mikroinstruktionen und enthält die *ComputerBlockController* der zugehörigen Computerblöcke sowie eine Map der *RegisterController* zur Ausführung der Operationen auf den Computerblöcken. Implementiert *IExecutor*.

⁴ggfs. werden private Hilfsmethoden nicht dargestellt

Attribute

Die Klasse stellt folgende Attribute mit Sichtbarkeit *private* bereit:

Name	Typ	Einschränkungen	Beschreibung
block- Controllers	List- <IQueryable- SimController>		Enthält die <i>BlockController</i> der zugehörigen Computerblöcke. Wird im Konstruktor gesetzt.
register- ControllerMap	Map<String, Register- Controller>		Hält die <i>RegisterController</i> für schnelleren Zugriff und Auflösung von String-Referenzen.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
execute	void	memory- Instruction: Memory- Instruction	Führt eine Speicheranweisung aus. Der Parameter <i>memoryInstruction</i> enthält den auszuführenden Speicherbefehl.
execute	void	conditioned- Instruction: Conditioned- Instruction	Führt eine bedingte Anweisung aus. Der Parameter <i>conditionedInstruction</i> enthält die auszuführende bedingte Anweisung.
execute	void	aluInstruction: AluInstruction	Führt eine Alu-Anweisung aus. Der Parameter <i>aluInstruction</i> enthält den auszuführenden Alu-Befehl.
execute	void	dataMovement- Instruction: DataMovement- Instruction	Führt eine Datenbewegungsanweisung aus. Der Parameter <i>dataMovementInstruction</i> enthält die auszuführende Datenbewegungsanweisung.

Zugehörigkeit

Die Klasse befindet sich im Paket `controller.simulation` und nutzt die Klassen `controller.blocks.-AluController`, `controller.blocks.BlockController`, `controller.blocks.BlockControllerType`, `controller.blocks.IQueryableSimController`, `controller.blocks.MemoryController`, `controller.blocks.RegisterController`, `model.instructionset.AluInstruction`, `model.instructionset.ConditionedInstruction`, `model.instructionset.DataMovementInstruction`, `model.instructionset.IExecutor`, `model.instructionset.MemoryInstruction`, `model.memoryrepresentation.Value`.

SimulationSequenceHandler $\langle CL8 \rangle$

Aufgabe

Steuert das Laden und Ausführen von Anweisungen im Computer. Wird vom `SimulationTimeHandler` aufgerufen, wenn der nächste Simulationstakt ausgeführt werden soll. Implementiert `Runnable`.

Attribute

Die Klasse stellt folgende Attribute mit Sichtbarkeit `private` bereit:

Name	Typ	Einschränkungen	Beschreibung
phaseCounter	int		Zählt die Anzahl der ausgeführten Anweisungen in der aktuellen Phase.
runPhase	RunPhase		Definiert die aktuelle Phase der Befehlausführung, entweder Abruf des Befehls oder Ausführung seiner Mikroinstruktionen.
micro-Instructions	IExecutable-Micro-Instruction[]		Enthält die Mikroinstruktionen des aktuellen Befehls.
block-Controllers	List-<IQueryable-SimController>		Enthält die <code>BlockController</code> der zugehörigen ComputerBlöcke. Wird als Konstruktorparameter gesetzt.

Name	Typ	Einschränkungen	Beschreibung
instructionSet-Model	IQuerable-InstructionSet-Model		Befehlssatzmodell, das alle Informationen darüber enthält, wie der Code auf der Grundlage des Befehlssatzes auszuführen ist.
program-Counter-Controller	Register-Controller		Controller des Befehlszählerregisters.
iarController	Register-Controller		Controller des Befehlsregisters.
memory-Controller	Memory-Controller		Controller des Speichers.
executor	Executor		<i>Executor</i> für die Mikroinstruktionen.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
run	void		Startet die Ausführung der aktuellen Anweisung. Wird vom <i>SimulationTimeHandler</i> aufgerufen, wenn der nächste Simulationstick ausgeführt wird. Beim ersten Aufruf der Ausführung einer neuen Anweisung wird <i>microInstructions</i> mit den erforderlichen Mikroanweisungen gefüllt. Anschließend und ansonsten wird die nächste Mikroinstruktion der aktuellen Anweisung ausgeführt.

Die Klasse stellt folgende Methoden mit Sichtbarkeit *private* bereit⁵:

⁵ggfs. werden private Hilfsmethoden nicht dargestellt

Name	Rückgabetyp	Parameter	Beschreibung
fetch	void		Führt einen Schritt der Abrufphase der Befehlsausführung aus. Die Abrufphase erfordert mehrere Schritte, die als Mikroinstruktionen dargestellt werden und im Befehlssatz definiert sind. Wenn der letzte Schritt der Abrufphase ausgeführt ist, wird die Ausführungsphase eingeleitet.
execute	void		Holt die aktuelle Mikroinstruktion aus <i>microInstructions</i> und führt sie aus. Für jede Mikroinstruktion ist ein Aufruf des <i>SimulationTimeHandler</i> notwendig. Jede Mikroinstruktion wird also einzeln ausgeführt.
executeMicroInstruction	void	instruction: IExecutableMicroInstruction	Ausführen einer Mikroinstruktion unter Verwendung eines Visitor-Patterns. Der Parameter <i>instruction</i> enthält die auszuführende Mikroinstruktion.

Zugehörigkeit

Die Klasse befindet sich im Paket *controller.simulation* und nutzt die Klassen *controller.blocks.BlockController*, *controller.blocks.BlockControllerType*, *controller.blocks.ControlUnitController*, *controller.blocks.IQueryableSimController*, *controller.blocks.MemoryController*, *controller.blocks.RegisterController*, *model.instructionset.IExecutableMicroInstruction*, *model.instructionset.IQueryableInstructionSetModel*.

SimulationTimeHandler`(CL9)`

Aufgabe

Handhabung des Timings der Simulationsausführung. Verwendet das Visitor-Pattern, um den Zustand des Taktgebers im Model zu verfolgen. Je nach Zustand wird der nächste Simulationstick ausgeführt oder die notwendige Wartezeit für die nächste Ausführung um einen Schritt verkürzt. Implementiert *ISimulationTimingObserver*.

Attribute

Die Klasse stellt folgende Attribute mit Sichtbarkeit *private* bereit:

Name	Typ	Einschränkungen	Beschreibung
executor-Service	ExecutorService		Steuert die Tick-Ausführung in einem neuen Thread.
simulation-Sequence-Handler	Simulation-Sequence-Handler		<i>SimulationSequenceHandler</i> , der den Simulationstick ausführt.
clockSpeed	int		Taktgeschwindigkeit für den Minecraft-Tick-Modus, die die Wartezeit zwischen den Ausführungen festlegt.
clockMode	ClockMode		Taktmodus, nach dem der Ausführungszeitpunkt bestimmt wird.
systemClock-Model	SystemClock-Model		<i>SystemClockModel</i> , das den Taktmodus festlegt.
minecraftTick-Counter	int		Zähler für den Minecraft-Tick-Modus, um die verbleibende Wartezeit zwischen den Ausführungen festzuhalten.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
onMinecraft-Tick	void		Wird vom <i>SystemClockController</i> aufgerufen, um den nächsten Simulationstick auszuführen. Wird ausgeführt, wenn der Minecraft-Tick-Modus aktiviert ist.
onUserTick-Trigger	void		Wird vom <i>SystemClockController</i> bei Benutzereingabe aufgerufen, um den nächsten Simulationstick auszuführen. Wird ausgeführt, wenn der Schrittmodus aktiviert ist.
onSimulation-TickComplete	void		Wird vom <i>SimulationSequenceHandler</i> aufgerufen, wenn ein Simulationsframe abgeschlossen ist. Wird ausgeführt, wenn der Echtzeitmodus aktiviert ist.
update-ObservedState	void		Hält den Tick-Modus unter Verwendung des Visitor-Patterns auf dem neuesten Stand.

Die Klasse stellt folgende Methoden mit Sichtbarkeit *private* bereit⁶:

⁶ggfs. werden private Hilfsmethoden nicht dargestellt

Name	Rückgabetyp	Parameter	Beschreibung
runTick	void		Stellt die Ausführung des nächsten Simulationsticks in die Warteschlange der Ausführungsthreads. Wirft eine <i>PreviousTickToSlowException</i> wenn der vornagegangene Tick noch nicht fertig ist. Wirft eine <i>SimulationErrorException</i> wenn im Simulationsschritt etwas schief läuft.

Zugehörigkeit

Die Klasse befindet sich im Paket *controller.simulation* und nutzt die Klassen *controller.blocks.-BlockController*, *controller.blocks.BlockControllerType*, *controller.blocks.IQueryableSimController*, *model.blocks.ClockMode*, *model.blocks.ISimulationTimingObserver*, *model.blocks.SystemClockModel*.

ClusterArchitectureHandler<CL10>

Aufgabe

Prüft ein Cluster auf eine valide Architekturen.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
check-Architecture	boolean	IArchitectureCheckable: IArchitectureCheckable, clusterHandler: ClusterHandler	static Prüft, ob es sich bei einem Cluster um eine bestimmte gültige Architektur handelt. Der Parameter <i>clusterHandler</i> enthält den <i>ClusterHandler</i> des zu prüfenden Clusters. Der Parameter <i>IArchitectureCheckable</i> enthält die Architektur auf die das Cluster geprüft werden soll.

Zugehörigkeit

Die Klasse befindet sich im Paket *controller.clustering*.

2.1.2 Model

Blocks

`BlockModel<CL11>`

Aufgabe

Diese Klasse repräsentiert das Model eines Blocks der Mod, welches die Daten für den zugehörigen Block hält. Jeder Block hat ein einzigartiges *BlockModel*. Implementiert *IControllerQueryableViewBlockModel* und *IViewQueryableViewBlockModel*.

Attribute

Die Klasse stellt folgende Attribute mit Sichtbarkeit *private* bereit:

Name	Typ	Einschränkungen	Beschreibung
hasUnqueried-StateChange	boolean		Wird auf true gesetzt, wenn eine Änderung der Daten vorliegt. Wird zurückgesetzt, wenn die View alle Daten übernommen hat.
type	ModelType		Typ des Models.
position	BlockPosition		Die Position des Minecraft-Blocks.

Operationen

Die Klasse stellt folgende Operationen mir Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
setPosition	void	position : BlockPosition	Setzt die Position des BlockModels.
getType	ModelType		Gibt den Typ des BlockModels zurück.
getPosition	BlockPosition		Gibt die Position des BlockModels zurück.

Die Klasse stellt folgende Operationen mir Sichtbarkeit *protected* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
setType	void	type: ModelType	Setzt den Typ dieses BlockModels.

Zugehörigkeit

Die Klasse befindet sich im Paket *model.blocks*.

Erbende Klassen

Von der Klasse erben folgende hier kurz mit Besonderheiten in der Funktionalität aufgelistete Klassen:

- *AluModel*: Hält die derzeitigen Operanden sowohl als auch die derzeitige Operation der ALU.
- *BusModel*: hält das zugehörige *BusSystemModel* $\langle CL28 \rangle$
- *RegisterModel*: Hält den derzeitigen Inhalt, den Register Typ und die architektur-spezifische Wortlänge.
- *SystemClockModel*: Hält den Modus, in welchem der Lerncomputer läuft, die Tick-Geschwindigkeit und eine Liste der Observer, welche den Modus beobachten.
- *ControlUnitModel*: Hält das *InstructionSetModel*.
- *MemoryModel*: Hält die eine *Memory*-Instanz, welche die Daten den Speicher des Lerncomputers beinhaltet. Mit den Methoden *getMemoryAt* und *setMemoryAt* lassen sich der Wert an einer bestimmten Adresse im Speicher Beschreiben und Auslesen.

Data

DataSetStringEntry $\langle CL12 \rangle$

Aufgabe

Diese Klasse stellt einen String-Eintrag im Datenbaum dar. Sie implementiert *IDataStringEntry*.

Attribute

Die Klasse stellt folgende Attribute mit Sichtbarkeit *private* bereit:

Name	Typ	Einschränkungen	Beschreibung
content	String		Der Inhalt dieses String-Eintrags.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
getContent	String		Ruft den Inhalt dieses String-Eintrags ab. Der Rückgabewert ist der Inhalt dieses Eintrags.
setContent	void	content: String	Setzt den Inhalt dieses String-Eintrags. Der Parameter <i>content</i> enthält den neuen Inhalt des Eintrags.
receive	void	visitor: IData-Visitor	Empfängt einen <i>IDataVisitor</i> und ruft die <i>visit</i> -Methode auf. Der Parameter <i>visitor</i> enthält den zu besuchenden Visitor.

Zugehörigkeit

Die Klasse befindet sich im Paket *model.data*.

Data(CL13)

Aufgabe

Diese Klasse ist ein Datencontainer. Sie wird verwendet, Daten zwischen View und Model/Controller auszutauschen. Sie implementiert *IDataContainer*.

Attribute

Die Klasse stellt folgende Attribute mit Sichtbarkeit *private* bereit:

Name	Typ	Einschränkungen	Beschreibung
contents	Map<String, IDataElement>		Der Inhalt dieses Datencontainers. Der Schlüssel der Map ist jeweils der Schlüssel, unter dem die Daten gespeichert sind. Der Wert ist das <i>IDataElement</i> selbst.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
set	void	key: String, value: IDataElement	Setzt den Wert eines Schlüssels. Der Parameter <i>key</i> enthält den Schlüssel, auf den die Daten gesetzt werden sollen. Der Parameter <i>value</i> enthält die Daten, die gesetzt werden sollen.

Name	Rückgabetyp	Parameter	Beschreibung
get	IDataElement	key: String	Ruft das Datenelement mit dem angegebenen Schlüssel ab. Der Parameter <i>key</i> enthält den Schlüssel, unter dem das Datenelement gespeichert ist. Der Rückgabewert ist das Datenelement. Gibt null zurück, wenn kein Datenelement unter dem angegebenen Schlüssel gespeichert ist.
receive	void	visitor: IData-Visitor	Empfängt einen <i>IDataVisitor</i> und ruft die visit-Methode auf. Der Parameter <i>visitor</i> enthält den zu benutzenden Visitor.
getKeys	Set<String>		Ruft die Schlüssel des Datencontainers ab. Der Rückgabewert ist der Schlüssel des Datencontainers.

Zugehörigkeit

Die Klasse befindet sich im Paket *model.data*.

InstructionSet

Instruction<CL14>

Aufgabe

Stellt eine Anweisung des Befehlssatzes dar. Implementiert *IQueryableInstruction*.

Attribute

Die Klasse stellt folgende Attribute mit Sichtbarkeit *private* bereit:

Name	Typ	Einschränkungen	Beschreibung
arguments	String[]		Argumente des Befehls.
opcode	String		Opcode des Befehls.
execution	Micro-Instruction[]		Ausführung des Befehls als eine Folge von Mikrobefehlen.
translation	String[]		Übersetzung des Befehls ins Binärformat.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
Instruction	Instruction	arguments: String[], opcode: String, execution: Micro-Instruction[], translation: String[]	Konstruktor
getOpcode	String		Getter für den Opcode des Befehls. Der Rückgabewert enthält den Opcode als String.
getTranslation	String[]		Getter für die binäre Übersetzung der Anweisung. Der Rückgabewert enthält die binäre Übersetzung der Anweisung als String-Array.

Name	Rückgabetyp	Parameter	Beschreibung
getArguments	String[]		Getter für die Argumente der Anweisung. Der Rückgabewert enthält die Argumente der Anweisung als String-Array.
getExecution	Micro-Instruction[]		Getter für die Ausführung der Anweisung. Der Rückgabewert enthält die Ausführungsvorschrift des Befehls als ein Array von <i>MicroInstructions</i> .

Zugehörigkeit

Die Klasse befindet sich im Paket *model.instructionset*.

InstructionCondition<CL15>**Aufgabe**

Stellt eine Bedingung in einer bedingten *MicroInstruction* dar.

Attribute

Die Klasse stellt folgende Attribute mit Sichtbarkeit *private* bereit:

Name	Typ	Einschränkung	Beschreibung
comparator	String		final Vergleichsoperation.
compare1	String		final Erster Vergleichswert.
compare2	String		final Zweiter Vergleichswert.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
get-Comparator	String		Getter für die Vergleichsoperation. Der Rückgabewert enthält die Vergleichsoperation als String.

Name	Rückgabetyp	Parameter	Beschreibung
getCompare1	String		Getter für den ersten Vergleichswert. Der Rückgabewert enthält den ersten Vergleichswert als String.
getCompare2	String		Getter für den zweiten Vergleichswert. Der Rückgabewert enthält den zweiten Vergleichswert als String.

Zugehörigkeit

Die Klasse befindet sich im Paket *model.instructionset*.

InstructionSetMemory $\langle CL16 \rangle$ **Aufgabe**

Modell für den Speicher des Befehlssatzes.

Attribute

Die Klasse stellt folgende Attribute mit Sichtbarkeit *private* bereit:

Name	Typ	Einschränkung	Beschreibung
wordLength	int		final Wortlänge in Bit.
addressLength	int		final Addresslänge in Bit.
accessDelay	int		final Zugriffsverzögerung in Takten.
byteOrder	String		final Bytereihenfolge als String.
possible- Opcode- Lengths	List<Integer>		final Mögliche Opcode-Längen als Liste von Integern.
opcodePosition	String		final Position des Opcodes.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
getWordSize	int		Getter für die Wortlänge. Der Rückgabewert enthält die Wortlänge in Bits.
getAddress-Size	int		Getter für die Addresslänge. Der Rückgabewert enthält die Adressgröße in Bits.
getAccess-Delay	int		Getter für die Zugriffsverzögerung. Der Rückgabewert enthält die Zugriffsverzögerung in Ticks.
getByteOrder	String		Getter für die Byte-Reihenfolge. Der Rückgabewert enthält die Byte-Reihenfolge als String.
getPossible-Opcode-Lengths	List<Integer>		Getter für die möglichen Opcode-Längen. Der Rückgabewert enthält die möglichen Opcode-Längen als Liste von Ganzzahlen.
getOpcode-Position	String		Getter für die Opcode-Position. Der Rückgabewert enthält die Opcode-Position als String.

Zugehörigkeit

Die Klasse befindet sich im Paket *model.instructionset*.

InstructionSetModel $\langle CL17 \rangle$

Aufgabe

Model eines Befehlssatzes. Enthält alle Informationen darüber, wie Code auf der Grundlage des Befehlssatzes ausgeführt werden kann. Implementiert *IQueryableInstructionSetModel*.

Attribute

Die Klasse stellt folgende Attribute mit Sichtbarkeit *private* bereit:

Name	Typ	Einschränkungen	Beschreibung
name	String		final Name des Befehlssatzes.
instruction- Length	int		final Länge der Anweisungen in Bits.
instructionSet- Registers	InstructionSet- Registers		final Registerspezifikationen des Befehlssatzes.
instructionSet- Memory	InstructionSet- Memory		final Speicherspezifikationen des Befehlssatzes.
aluActions	String[]		final ALU-Operationen des Befehlssatzes.
fetchPhase	Micro- Instruction[]		final Spezifikation der Hol-Phase für den Befehlssatz.
address- ChangeHash- Map	HashMap<- String, String>		final Optionen zum Ändern der Adresse, an der bestimmte Teile eines Programms gespeichert werden sollen.
programStart- Label	String		final Label, an dem das Programm standardmäßig gestartet werden soll.
dataStorage- Keywords	HashMap<- String, String>		final Schlüssel zur Angabe der Datenspeicherungsarten.
command- HashMap	HashMap<- String, Instruction>		final Befehle des Befehlssatzes, abgebildet nach Befehlsschlüssel.
opcodeHash- Map	HashMap<- String, Instruction>		final Befehle des Befehlssatzes, abgebildet nach Opcode.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
generate-Opcode-Hashmap	void		Erzeugt die Opcode-Hashmap aus der Befehlshashmap.
getAlu-Registers	String[]		Getter für die Register des Befehlssatzes.
getName	String		Getter für den Namen des Befehlssatzes.
getInteger-Register	Integer	key: String	Getter für die Integer-Register des Befehlssatzes. Der Parameter <i>key</i> enthält den Schlüssel des Registers. Der Rückgabewert enthält das Integer-Register, das dem Schlüssel entspricht.
getProgram-Counter	String		Getter für das Programmzählerregister des Befehlssatzes.
getMemory-WordSize	int		Getter für die Wortlänge des Befehlssatzspeichers.
getMemory-AddressSize	int		Getter für die Adressgröße des Befehlssatzspeichers.
getInstruction	Instruction	key: String	Getter für eine Anweisung, die durch ihr Befehlsschlüsselwort angegeben wird. Der Parameter <i>key</i> enthält das Befehlsschlüsselwort des Befehls. Der Rückgabewert gibt die Anweisung zurück, die dem Befehlsschlüsselwort entspricht.

Name	Rückgabetyp	Parameter	Beschreibung
isAddress-Change	boolean	string: String	Prüft, ob ein String der Adressänderungsspezifikation des Befehlssatzes entspricht. Der Parameter <i>string</i> enthält den zu prüfenden String. Der Rückgabewert ist true, wenn der String der Adressänderungsspezifikation des Befehlssatzes entspricht, andernfalls false.
getChanged-Address	String	label: String	Löst ein Adressänderungslabel in eine Adresse auf. Der Parameter <i>label</i> enthält das Adressänderungslabel. Der Rückgabewert enthält die Adresse, die mit dem Label übereinstimmt.
getProgram-StartLabel	String		Getter für das Programmstart-Label.
isDataStorage-Command	boolean	string: String	Prüft, ob ein String ein Datenspeicherbefehl ist. Der Parameter <i>string</i> enthält den zu prüfenden String. Der Rückgabewert ist true, wenn der String ein Datenspeicherbefehl ist, andernfalls false.
getStorage-Command-Data	String	string: String	Löst einen Datenspeicherbefehl auf in die Daten in dem Format, in dem sie gespeichert werden sollen. Der Parameter <i>string</i> enthält den Datenspeicherungsbefehl. Der Rückgabewert liefert die zu speichernden Daten.

Name	Rückgabetyp	Parameter	Beschreibung
getFetch-PhaseLength	int		Getter für die Hol-Phasenlänge des Befehlssatzes.
getFetch-PhaseStep	Micro-Instruction	index: int	Getter für eine <i>MicroInstruktion</i> der Hol-Phase des Befehlssatzes. Der Parameter <i>index</i> enthält den Index der <i>MikroInstruktion</i> in der Abrupphase. Der Rückgabewert enthält die <i>MikroInstruktion</i> am angegebenen Index.

Zugehörigkeit

Die Klasse befindet sich im Paket *model.instructionset*.

InstructionSetRegisters⟨CL18⟩**Aufgabe**

Modell der Befehlssatzregister. Enthält alle Informationen zu den Registern des Befehlsatzes.

Attribute

Die Klasse stellt folgende Attribute mit Sichtbarkeit *private* bereit:

Name	Typ	Einschränkungen	Beschreibung
program-Counter	String		final Name des Programmzählerregisters.
aluRegs	String[]		final ALU-Registernamen.
floatRegs	HashMap<- String, Integer>		final Zuordnung von Adressen und Namen zu Fließkommaregistern.
intRegs	HashMap<- String, Integer>		final Integer-Registerabbildung von Adressen und Namen.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
getFloat-Register	Integer	name: String	Getter für die Fließkommaregister. Der Parameter <i>name</i> enthält den Namen des Registers. Der Rückgabewert gibt die Adresse des Registers zurück.
getInteger-Register	Integer	name: String	Getter für die Integer-Register. Der Parameter <i>name</i> enthält den Namen des Registers. Der Rückgabewert gibt die Adresse des Registers zurück.
getProgram-Counter	String		Getter für das Programmzählerregister.
getAluRegs	String[]		Getter für die ALU-Register.

Zugehörigkeit

Die Klasse befindet sich im Paket *model.instructionset*.

MemoryInstruction<CL19>

Aufgabe

Stellt eine Speicheraktion als Teil einer Mikroanweisung dar. Erbt von *MicroInstruction*.

Attribute

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
execute	void	executor: IExecutor	Übergibt die Speicheraktion an den <i>Executor</i> und führt sie als Teil eines Visitor-Patterns aus.

Zugehörigkeit

Die Klasse befindet sich im Paket *model.instructionset*.

MicroInstruction<CL20>

Aufgabe

abstract Die Klasse hält gemeinsame Attribute und Funktionen von Mikrobefehlen.

Implementiert das Interface *IExecutableMicroInstruction*.

Attribute

Name	Typ	Einschränkungen	Beschreibung
from	String[]		final Das oder die Register, aus denen gelesen werden soll.
to	String		final Das Register, in welches geschrieben werden soll.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
getFrom	String[]		Getter-Methode für die Register, aus denen gelesen werden soll.
getTo	String		Getter-Methode für das Register, in welches geschrieben werden soll.

Zugehörigkeit

Die Klasse befindet sich im Paket *model.instructionset*.

Erbende Klassen

Von der Klasse erbt die Klasse *ComplexMicroInstruction <CL23>*.

MicroInstructionsDeserializer<CL21>

Aufgabe

Deserializer für *MicroInstructions*, um das Json-Parsing an das Json-Format des Befehlsatzes anzupassen.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
deserialize	MicroInstruction	jsonElement: JsonElement, type: Type, json- Deserialization- Context: Json- Deserialization- Context	Deserialisiert ein Json-Element in eine <i>MicroInstruction</i> . Wenn das Json-Element nicht gelesen werden konnte, wird eine <i>JsonParseException</i> geworfen.

Die Klasse stellt folgende Methoden mit Sichtbarkeit *private* bereit⁷:

Name	Rückgabetyp	Parameter	Beschreibung
parseJson- Array	Micro- Instruction	jsonArray: Json- Array	Parse eines JSONArray in eine <i>MicroInstruction</i> bei gleichzeitiger Unterscheidung der <i>MicroInstruction</i> -Typen. Der Parameter <i>jsonArray</i> enthält eine <i>MicroInstruction</i> in einem der drei möglichen Formate. Der Rückgabewert gibt eine <i>MicroInstruction</i> -Instanz zurück, die dem Anweisungstyp entspricht.
parseData- Movement- Instruction	DataMovement- Instruction	jsonArray: Json- Array	Erzeugt <i>DataMovementInstructions</i> aus json-Arrays der Form [“<destination>”, “<origin>”, “<memory flag>”, “<memory microinstruction>”].

⁷ggfs. werden private Hilfsmethoden nicht dargestellt

Name	Rückgabetyp	Parameter	Beschreibung
parseMemory-Instruction	Memory-Instruction	jsonElement: JsonElement	Erzeugt <i>MemoryInstructions</i> aus json-Arrays der Form [“<destination>”, “<origin>”].
parseAlu-Instruction	AluInstruction	jsonArray: Json-Array	Erzeugt <i>AluInstruction</i> aus json der Form [“<operation>”, “<alu-dest>”, “<alu-origin 1>”, “<alu-origin 2>”, “<memory flag>”, “<memory microinstruction>”].
parse-Conditioned-Instruction	Conditioned-Instruction	jsonArray: Json-Array	Erzeugt <i>ConditionedInstructions</i> aus json-Arrays der Form [“IF”, “[“<comparator 1>”, “<comparator 2>”, “<comparing operation>”], [“<then destination>”, “<then origin>”], “<memory flag>”, “<memory microinstruction>”].
parse-Instruction-Condition	Instruction-Condition	jsonArray: Json-Array	Erzeugt <i>InstructionConditions</i> aus json-Arrays der Form [“<comparator 1>”, “<comparator 2>”, “<comparing operation>”].

Zugehörigkeit

Die Klasse befindet sich im Paket *model.instructionset*.

AluInstruction<CL22>

Aufgabe

Hält eine Alu-Anweisung wie eine arithmetische Operation. Erbt von der *ComplexMicro-Instruction*.

Attribute

Die Klasse stellt folgende Attribute mit Sichtbarkeit *private* bereit:

Name	Typ	Einschränkungen	Beschreibung
AluInstruction	AluInstruction	from: String[], to: String, memoryFlag: String, memory- Instruction: Memory- Instruction, action: String	Konstruktor
action	String		Alu-Aktion der Anweisung.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
getAction	String		Gibt die Alu-Aktion der Anweisung zurück.
execute	void	executor: IExecutor	Übergibt die Alu-Anweisung an den <i>Executor</i> und führt sie als Teil eines Visitor-Patterns aus.

Zugehörigkeit

Die Klasse befindet sich im Paket *model.instructionset*.

ComplexMicroInstruction<CL23>

Aufgabe

abstract Die Klasse hält gemeinsame Attribute und Funktionalitäten komplexer *MicroInstructions*, die ein Speicher-Flag und einen Speicher-Befehl enthalten können. Die Klasse erbt von *MicroInstruction* <CL20>.

Attribute

Die Klasse stellt folgende Attribute mit Sichtbarkeit *private* bereit:

Name	Typ	Einschränkungen	Beschreibung
memoryFlag	String		final Speicherflag für die Art des Speicherzugriffs. Wird im Constructor gesetzt.
memory-Instruction	Memory-Instruction		final Speicheranweisung für parallelen Speicherzugriff. Wird im Constructor gesetzt.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
getMemory-Flag	String		Getter für die Speicherflag.
getMemory-Instruction	Memory-Instruction		Getter für die Speicheranweisung.

Zugehörigkeit

Die Klasse befindet sich im Paket *model.instructionset*.

Erbende Klassen

Von der Klasse erben folgende Klassen:

- *AluInstruction* <CL22>
- *ConditionedInstruction* <CL24>
- *DataMovementInstruction* <CL25>

ConditionedInstruction $\langle CL24 \rangle$

Aufgabe

Enthält eine konditionierte Anweisung wie einen Sprung oder eine Verzweigung. Erbt von *ComplexMicroInstruction*.

Attribute

Die Klasse stellt folgende Attribute mit Sichtbarkeit *private* bereit:

Name	Typ	Einschränkungen	Beschreibung
condition	Instruction-Condition		Bedingung der Anweisung.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
execute	void	executor: I-Executor	Übergibt die konditionierte Anweisung an den <i>Executor</i> und führt sie als Teil eines Visitor-Patterns aus.
getCondition	Instruction-Condition		Getter für die Bedingung der Anweisung. Der Rückgabewert gibt die Bedingung der Anweisung zurück.

Zugehörigkeit

Die Klasse befindet sich im Paket *model.instructionset*.

DataMovementInstruction $\langle CL25 \rangle$

Aufgabe

Enthält eine Datenbewegungsanweisung, wie eine Registerlade- oder Speicheroperation.

Erbt von *ComplexMicroInstruction*.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
execute	void	executor: I-Executor	Übergibt die Datenbewegungs-Anweisung an den <i>Executor</i> und führt sie als Teil eines Visitor-Patterns aus.

Zugehörigkeit

Die Klasse befindet sich im Paket *model.instructionset*.

Sonstige Model-Klassen

Memory $\langle CL26 \rangle$

Aufgabe

Diese Klasse stellt den Speicher des Computers dar. Sie wird verwendet, um *Values* an Adressen zu speichern. Eine Speicherinstanz hat eine feste Wort- und Adressgröße. Ein Speicher kann beschrieben und gelesen werden.

Attribute

Die Klasse stellt folgende Attribute mit Sichtbarkeit *private* bereit:

Name	Typ	Einschränkungen	Beschreibung
memory	Map<Value, Value>		final Der eigentliche Speicher. Eine Zuordnung von Adressen zu Werten.
memory-Length	int		final Die Größe eines Speicherworts in Bytes. Wird im Constructor gesetzt.

Name	Typ	Einschränkungen	Beschreibung
addressLength	int		final Die Größe einer Speicheradresse in Bytes. Wird im Constructor gesetzt.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
getValueAt	Value	address: Value	Gibt den Wert an der angegebenen Adresse zurück. Der Parameter <i>address</i> enthält die Adresse, von der gelesen werden soll. Der Rückgabewert gibt den Wert an der angegebenen Adresse zurück. Gibt einen neuen <i>Value</i> mit der Größe des Speicherworts zurück und Inhalt 0, wenn die Adresse nicht festgelegt ist.
setValue	void	address: Value, value : Value	Setzt den Wert an der angegebenen Adresse. Der Parameter <i>address</i> enthält die Adresse, an die geschrieben werden soll. Der Parameter <i>value</i> enthält den zu schreibenden Wert.
getData	IDataElement		Speichert den Speicher in einem <i>IDataElement</i> . Gibt den gespeicherten Speicher zurück.
static fromData	Memory	data: IData-Container	Lädt einen Speicher aus einem <i>IDataContainer</i> . Der Parameter <i>data</i> enthält den <i>IDataContainer</i> , aus dem geladen werden soll. Der Rückgabewert gibt den geladenen Speicher zurück.

Zugehörigkeit

Die Klasse befindet sich im Paket *model.memoryrepresentation* und nutzt die Klassen *model.data.IDataContainer*, *model.data.IDataElement*.

Value(CL27)

Aufgabe

Diese Klasse stellt einen Wert dar. Sie bietet Methoden zur Konvertierung des Wertes in und aus dem Binär- und Hexadezimalsystem, als auch für Dezimal und Fließkomma-Zahlen. Erlaubt die Inkrementierung des Wertes (als Kopie).

Attribute

Die Klasse stellt folgende Attribute mit Sichtbarkeit *private* bereit:

Name	Typ	Einschränkungen	Beschreibung
value	byte[]		Enthält den Wert als Byte-Array.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
fromHex	Value	string: String, length: int	static Erzeugt einen Wert aus einer hexadezimalen Zeichenkette. Der Parameter <i>string</i> enthält die hexadezimale Zeichenfolge. Der Parameter <i>length</i> gibt die Länge des Wertes in Bytes an. Der Rückgabewert gibt den Wert zurück.

Name	Rückgabetyp	Parameter	Beschreibung
fromBinary	Value	string: String, length: int	static Erzeugt einen Wert aus einer binären Zeichenkette. Der Parameter <i>string</i> enthält die binäre Zeichenfolge. Der Parameter <i>length</i> gibt die Länge des Wertes in Bytes an. Der Rückgabewert gibt den Wert zurück.
fromFloat	Value	string: String, length: int	static Erzeugt einen Wert aus einer dezimalen Fließkomma-Zeichenkette. Der Parameter <i>string</i> enthält die Float-Zeichenkette. Der Parameter <i>length</i> gibt die Länge des Wertes in Bytes an. Der Rückgabewert gibt den Wert zurück.
fromDecimal	Value	string: String, length: int	static Erzeugt einen Wert aus einer dezimalen Zeichenkette. Der Parameter <i>string</i> enthält die Int-Zeichenkette. Der Parameter <i>length</i> gibt die Länge des Wertes in Bytes an. Der Rückgabewert gibt den Wert zurück.
getByteValue	byte[]		Gibt den Wert als Byte-Array zurück.
getBinary-Value	String		Gibt den Wert als binäre Zeichenkette zurück.
getFloatValue	String		Gibt den Wert als Fließkomma-Dezimal-String zurück.
get-Hexadecimal-Value	String		Gibt den Wert als Hexadezimal-String zurück.

Name	Rückgabetyp	Parameter	Beschreibung
get-Incremented-Value	Value		Gibt einen um 1, von diesem Wert, inkrementierten Wert zurück
equals	boolean	o: Object	Vergleicht Werte anhand ihrem <i>value</i> .
hashCode	int		Rechnet den Hashwert von <i>value</i> aus und gibt ihn zurück.

Zugehörigkeit

Die Klasse befindet sich im Paket *model.memoryrepresentation*.

BusSystemModel<CL28>**Aufgabe**

Diese Klasse repräsentiert ein zusammenhängendes Netz aus *BusBlöcken*, also die Graph-Repräsentation eines Lerncomputers. Implementiert *IQueryableBusSystem*.

Attribute

Die Klasse stellt folgende Attribute mit Sichtbarkeit *private* bereit:

Name	Typ	Einschränkungen	Beschreibung
adjPositions	Map<Block-Position, List<Block-Position>>		Graph-Repräsentation aller Blöcke im Cluster und ihrer Verbindungen.
presentData	Value		Die auf dem Bus-System liegenden Daten.
from	BlockPosition		Zeigt, von welchem Block die aktuellen Daten kommen.
to	BlockPosition		Zeigt, zu welchem Block die aktuellen Daten gehen.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
getPresent-Data	Value		Gibt die aktuell auf dem Bus vorliegenden Daten zurück.
getActive-Visualization	boolean	blockPosition: BlockPosition	Gibt zurück, ob über den Bus gerade aktiv Daten fließen.
addNode	void	newBlock: Block-Position	Fügt dem Graphen des <i>BusSystemModels</i> einen neuen Knoten hinzu. Der Parameter <i>newBlock</i> enthält den neuen Knoten.
addEdge	void	pos1: Block-Position, pos2: BlockPosition	Fügt eine Kante zwischen zwei Knoten ein. Der Parameter <i>pos1</i> ist der erste Knoten der Kante. Der Parameter <i>pos2</i> ist der zweite Knoten der Kante.
removeEdge	void	pos1: Block-Position, pos2: BlockPosition	Entfernt eine Kante zwischen zwei Knoten. Der Parameter <i>pos1</i> ist der erste Knoten der Kante. Der Parameter <i>pos2</i> ist der zweite Knoten der Kante.
removeNode	void	pos1: Block-Position	Entfernt einen Knoten und alle mit ihm verbundenen Kanten. Der Parameter, den <i>pos1</i> enthält, ist der zu entfernende Knoten.
getBusGraph	Map<Block-Position, List<Block-Position>>		Gibt den Graphen des <i>BusSystemModels</i> zurück. Der Rückgabewert gibt das Attribut <i>adjPositions</i> zurück.

Name	Rückgabetyp	Parameter	Beschreibung
combineGraph	void	ownNode : BlockPosition, newNode : BlockPosition, busSystem- ToCombine : IQueryableBus- System	Verbindet zwei Graphen und speichert sie in der aktuellen Instanz.
splitBus- SystemModel	List<Bus- SystemModel>	deletedNode: BlockPosition	Entfernt einen Knoten und alle mit ihm verbundenen Kanten. Falls der Graph nicht mehr zusammenhängt, werden neue <i>BusSystemModels</i> erstellt.
isNode	boolean	blockPosition: BlockPosition	Prüft, ob ein Knoten im BusSystemModel vorhanden ist. Der Parameter <i>blockPosition</i> enthält den zu prüfenden Knoten. Der Rückgabewert gibt true zurück, wenn der Knoten im BusSystemModel ist.

Die Klasse stellt folgende Methoden mit Sichtbarkeit *private* bereit⁸:

Name	Rückgabetyp	Parameter	Beschreibung
bfs	Map<Block- Position, List<Block- Position>>	start: Block- Position	BFS, um alle verbundenen Knoten zu finden. Der Parameter <i>start</i> enthält den Startknoten. Der Rückgabewert gibt den Graph der mit dem Startknoten verbundenen Knoten zurück.

⁸ggfs. werden private Hilfsmethoden nicht dargestellt

Name	Rückgabetyp	Parameter	Beschreibung
splitGraph	List<Map<-BlockPosition, List<Block-Position>>>	deletedNode: BlockPosition	Entfernt einen Knoten und seine Kanten und teilt den Graphen bei Bedarf auf. Der Parameter <i>deletedNode</i> enthält den zu entfernenden Knoten. Der Rückgabewert gibt die Liste der neuen Graphen zurück.

Zugehörigkeit

Die Klasse befindet sich im Paket *model.busgraph* und nutzt die Klassen *model.memory-representation.Value*, *model.blocks.BlockPosition*.

2.1.3 View**Client****ProgrammingScreen**⟨CL29⟩**Aufgabe**

Diese Klasse stellt den Programmierbildschirm dar. Sie wird geöffnet, wenn ein Spieler einen Programmierblock verwendet. Sie wird geschlossen, wenn der Spieler den Bildschirm schließt. Sie bietet ein Textfeld zur Eingabe des Codes. Die Klasse erbt von *HandledScreen*⟨*ProgrammingScreenHandler*⟩ Wird hier exemplarisch für alle Screens der Mod (ControlUnitScreen, MemoryScreen, RegisterScreen und SystemClockScreen) vorgestellt.

Attribute

Die Klasse stellt folgende Attribute mit Sichtbarkeit *private* bereit:

Name	Typ	Einschränkungen	Beschreibung
editBox	EditBoxWidget		Das Editierfeld-Widget, das für die Eingabe des Codes verwendet wird.
assemble-Button	ButtonWidget		Die Schaltfläche, die zum Assembeln des Codes verwendet wird.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *protected* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
init	void		Initialisiert den Bildschirm. Fügt das Bearbeitungsfeld-Widget zum Bildschirm hinzu. Fügt die "assemble" Schaltfläche zum Bildschirm hinzu.
draw-Background	void	context: Draw-Context, float: , int: , int:	Zeigt den Hintergrund des Bildschirms an. Der Parameter <i>context</i> enthält den Mal-Kontext. Der Parameter <i>mouseX</i> gibt die x-Position der Maus an. Der Parameter <i>mouseY</i> gibt die y-Position der Maus an. Der Parameter <i>delta</i> gibt das Zeitdelta seit der letzten Bildschirmaktualisierung an.

Zugehörigkeit

Die Klasse befindet sich im Paket `view.client.screens.handled` und nutzt die Klasse `view.main.blocks.mod.programming.ScreenHandler`.

RenderMixin(*CL30*)

Aufgabe

Diese Klasse wird beim Minecraft-Start in den Bildschirm-Aktualisierungs-Teil Minecraft's *InGameHud* Klasse eingefügt. Dies erlaubt es uns etwas auf dem Bildschirm darzustellen und somit die *GoggleUI* anzuzeigen.

Attribute

Die Klasse stellt folgende Attribute mit Sichtbarkeit *private* bereit:

Name	Typ	Einschränkungen	Beschreibung
gui	GoggleUI		Die <i>GoggleUI</i> -Instanz.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *private* bereit⁹:

Name	Rückgabetyp	Parameter	Beschreibung
beforeRender-DebugScreen	void	context: DrawContext, f : float, ci : CallbackInfo	Diese Methode wird in das Minecraft Render System eingefügt. Sie wird ausgeführt, wenn Minecraft den Bildschirm neu lädt. Ruft die <i>GoggleUI</i> auf, welche Informationen zum Block anzeigt, der gerade betrachtet wird. Der Parameter <i>context</i> enthält den Zeichenkontext. Der Parameter <i>f</i> ist die Zeit seit der letzten Bildschirmaktualisierung. Der Parameter <i>ci</i> enthält Informationen zum Aufruf der originalen Methode.

Zugehörigkeit

Die Klasse befindet sich im Paket *view.client.mixin* und nutzt die Klasse *view.client.renderlistener.-GoggleUI*.

GoggleUI(CL31)

Aufgabe

final Diese Klasse ist für das Darstellen der Benutzeroberfläche verantwortlich, wenn der Spieler einen Block betrachtet, der *IGoggleQueryable* implementiert.

Attribute

Die Klasse stellt folgende Attribute mit Sichtbarkeit *private* bereit:

Name	Typ	Einschränkungen	Beschreibung
visible	boolean		Gibt an, ob die UI sichtbar ist.

⁹ggfs. werden private Hilfsmethoden nicht dargestellt

Name	Typ	Einschränkungen	Beschreibung
textRenderer	TextRenderer		final Der Text-Renderer für die UI.
minecraft	MinecraftClient		final Der Minecraft-Client, auf dem die UI dargestellt wird.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
onRender-GameOverlay	void	context: Draw-Context	Methode, die bei jedem Frame aufgerufen wird. Rendert die Benutzeroberfläche, wenn der Spieler auf einen Block blickt, der <i>IGoggleQueryable</i> implementiert. Der Parameter <i>context</i> enthält den Zeichenkontext.

Die Klasse stellt folgende Methoden mit Sichtbarkeit *private* bereit¹⁰:

Name	Rückgabetyp	Parameter	Beschreibung
drawUI	void	context: Draw-Context, text : Text	Zeichnet die Benutzeroberfläche. Der Parameter <i>context</i> enthält den Zeichenkontext. Der Parameter <i>text</i> enthält den zu zeichnenden Text.

Zugehörigkeit

Die Klasse befindet sich im Paket *view.client.renderlistener* und nutzt die Klasse *view.main.blocks.mod.com.IGoggleQueryable*.

¹⁰ggfs. werden private Hilfsmethoden nicht dargestellt

Main

Blocks

Mod **ModBlock***<CL32>*

Aufgabe

Diese Klasse repräsentiert einen Block aus unserem Mod im Spiel. Jeder Block des gleichen Typs in der Welt ist eine Klasse. Sie erbt von der Minecraft Klasse *BlockWithEntity*.

Attribute

Die Klasse stellt folgende Attribute mit Sichtbarkeit *private* bereit:

Name	Typ	Einschränkungen	Beschreibung
BLOCK_- STRENGTH	float		Standardstärke aller Blöcke.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
onUse	ActionResult	state: BlockState, world: World, pos: BlockPos, player: PlayerEntity, hand: Hand, hit: BlockHitResult	Diese Methode wird aufgerufen, wenn der Block mit der rechten Maustaste angeklickt wird. Wenn der Block ein GUI bietet, wird dieses geöffnet. Der Parameter <i>state</i> enthält den Blockstatus. Der Parameter <i>world</i> enthält die Minecraft-Welt, in der der Block platziert ist. Der Parameter <i>pos</i> enthält die Position, an der der Block platziert ist. Der Parameter <i>player</i> enthält den Spieler, der mit der rechten Maustaste auf den Block geklickt hat. Der Parameter <i>hand</i> enthält die Hand, mit der der Spieler den Block mit der rechten Maustaste angeklickt hat. Der Parameter <i>hit</i> enthält das Trefferergebnis des Raycasts. Der Rückgabewert gibt <i>ActionResult.SUCCESS</i> zurück, wenn der Block einen Screenhandler hat oder die Welt clientseitig ist, <i>ActionResult.PASS</i> andernfalls.

Name	Rückgabetyp	Parameter	Beschreibung
onState-Replaced	void	state : BlockState, world : World, pos : BlockPos, newState : BlockState, moved : boolean	Wird aufgerufen, wenn sich der Zustand des <i>ModBlocks</i> verändert. Wenn der Block zerstört wurde, werden mögliche Inhalte auf zurückgelassen. Der Parameter <i>state</i> enthält den alten Blockstatus. Der Parameter <i>world</i> enthält die Minecraft-Welt, in der der Block platziert ist. Der Parameter <i>pos</i> gibt die Position des Blocks an. Der Parameter <i>newState</i> enthält den neuen Status des Blocks. Der Parameter <i>moved</i> enthält true, wenn der Block verschoben wurde, sonst false.
getRender-Type	BlockRender-Type	state: BlockState	Gibt den Rendertyp (in der Regel <i>BlockRenderType.MODEL</i>) des Blocks zurück. Der Parameter <i>state</i> enthält den Blockstatus.

Zugehörigkeit

Die Klasse befindet sich im Paket *view.main.blocks.mod*

Erbende Klassen

Von der Klasse erben der abstrakte ComputerBlock (CL36) und der ProgrammingBlock.

ModBlockEntity<CL33>

Aufgabe

abstract Diese Klasse repräsentiert eine *BlockEntity* aus unserer Mod im Spiel. Jeder Block hat seine eigene, einzigartige *BlockEntity*, während er von Minecraft geladen ist. Die Klasse erbt von der Minecraft-Klasse *BlockEntity*.

Attribute

Die Klasse stellt folgende Attribute mit Sichtbarkeit *private* bereit:

Name	Typ	Einschränkungen	Beschreibung
CONTROLLER NBT _ TAG	_ - String		static final Speicherort der Daten für diesen Block.
controller	IUserInput- Receivable- Controller		Der Controller dieser <i>BlockEntity</i> . Kann null sein, wenn der Block gerade geladen wird. Verwaltet die Computerlogik des Blocks und die Kommunikation mit dem Model. Ist auf der Client-Seite null.
entitytype	EntityType		Der Typ dieser <i>ModBlockEntity</i> . Wird verwendet, um zwischen verschiedenen Arten von <i>ModBlockEntitys</i> zu unterscheiden.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
setController	void		Wird aufgerufen, wenn der Block in der Welt platziert wird. Sie erstellt den Controller für den Block, wenn sie aus dem Serverkontext aufgerufen wird.

Name	Rückgabetyp	Parameter	Beschreibung
readNbt	void	nbt: Nbt-Compound	Liest Daten aus dem Named Binary Tag (NBT) und speichert sie in der Entity. Gibt sie auch an den eigenen <i>BlockController</i> weiter.
writeNbt	void	nbt: Nbt-Compound	Speichert Daten aus der <i>ModBlockEntity</i> im NBT.
getBlock-Position	BlockPosition		Getter für die Position dieser <i>ModBlockEntity</i> .
getModblock-Type	EntityType		Getter für den Typ dieser <i>ModBlockEntity</i> .
getController	IUserInput-Receivable-Controller		Gibt den Controller dieser <i>ModBlockEntity</i> zurück. Im Client-Kontext gibt die Methode null zurück.
setType	void	type: EntityType	Legt den Typ dieser <i>ModBlockEntity</i> fest.

Die Klasse stellt folgende Methoden mit Sichtbarkeit *protected* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
create-Controller	IUserInput-Receivable-Controller		abstract Jede <i>ModBlockEntity</i> benötigt einen eigenen, benutzerdefinierten <i>BlockController</i> . Gibt den Controller dieser <i>ModBlockEntity</i> zurück.

Zugehörigkeit

Die Klasse befindet sich im Paket *view.main.blocks.mod* und nutzt die Klassen *controller.blocks.-IUserInputReceivableController*, *model.blocks.BlockPosition*, *view.main.data.NbtDataConverter*.

Erbende Klassen

Von der Klasse erben die abstrakten Klassen *ComputerBlockEntity* *(CL37)* und *ModBlockEntityWithInventory*.

ModBlockEntityWithInventory(*CL34*)

Aufgabe

abstract Diese Klasse repräsentiert eine *BlockEntity*, die ein Minecraft-Inventar hat. Ein Inventar ermöglicht es dem Block, Gegenstände zu speichern. Die Klasse erbt von der *ModBlockEntity* und implementiert *ImplementedInventory*.

Attribute

Die Klasse stellt folgende Attribute mit Sichtbarkeit *protected* bereit:

Name	Typ	Einschränkungen	Beschreibung
items	DefaultedList<- ItemStack>		Diese Liste enthält alle Gegenstände, die im Inventar gespeichert sind.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
getItems	DefaultedList<- ItemStack>		Gibt die Liste der im Inventar gespeicherten Gegenstände zurück.

Zugehörigkeit

Die Klasse befindet sich im Paket *view.main.blocks.mod*.

Erbende Klassen

Von der Klasse erbt die *ProgrammingBlockEntity* und die *ComputerBlockEntityWithInventory*.

ModScreenHandler $\langle CL35 \rangle$

Aufgabe

abstract Klasse die grundlegende *ScreeenHandler* Funktionalitäten bietet, die für mehrere *ScreeenHandler* gleich wären. Erbt von *ScreeenHandler*.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
quickMove	ItemStack	player: Player-Entity, slot : int	Bewegt einen <i>ItemStack</i> , bei Inventar-Schnell-Bewegungen.
canUse	boolean	player: Player-Entity	Bestimmt ob der Spieler mit diesem ScreenHandler interagieren kann.

Zugehörigkeit

Die Klasse befindet sich im Paket *view.main.blocks.mod*.

Erbende Klassen

Von der Klasse erben folgende hier kurz mit Besonderheiten in der Funktionalität aufge-listete Klassen:

- *ControlUnitScreenHandler* Bietet Kommunikation zwischen Client und Server für den *ControlUnitScreen*.
- *RegisterScreenHandler* Bietet Kommunikation zwischen Client und Server für den *RegisterScreen*.
- *SystemClockScreenHandler* Bietet Kommunikation zwischen Client und Server für den *SystemClockScreen*.
- *ProgrammingScreenHandler* Bietet Kommunikation zwischen Client und Server für den *ProgrammingScreen*.
- *MemoryScreenHandler* Bietet Kommunikation zwischen Client und Server für den *MemoryScreen*.

Computer ComputerBlock<CL36>

Aufgabe

abstract Umfasst alle *ModBlocks*, die zum Bau eines Computers benötigt werden. Diese Blöcke können über einen *BusBlock* verbunden werden oder sind ein *BusBlock*, um einen funktionierenden Computer zu bilden. Jeder *ComputerBlock* hat zur Laufzeit eine eindeutige *ComputerBlockEntity*. Erbt von *ModBlock*.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
onState-Replaced	void	state: BlockState, world : World , pos : BlockPos , newState Block- State , moved : boolean	Wird bei jeder Änderung des Blockzustands aufgerufen. Wird verwendet, um der <i>BlockEntity</i> mitzuteilen, dass der zugehörige Block zerstört wurde. Der Parameter <i>state</i> enthält den alten Blockstatus. Der Parameter <i>world</i> enthält die Minecraft-Welt, in der der Block platziert ist. Der Parameter <i>pos</i> gibt die Position des Blocks an. Der Parameter <i>newState</i> enthält den neuen Status des Blocks. Der Parameter <i>moved</i> enthält true, wenn der Block verschoben wurde, sonst false.
getTicker	<T extends BlockEntity> BlockEntityTi- cker<T>	world : World, state : Block- State, type : BlockEntity- Type<T>	Informiert Minecraft darüber, dass ticks für diesen Block durch die <i>ComputerBlockEntity</i> -Methode <i>tick()</i> verarbeitet werden.

Zugehörigkeit

Die Klasse befindet sich im Paket *view.main.blocks.mod.computer* und nutzt die Klassen *view.main.blocks.mod.ModBlock*, *view.main.blocks.mod.computer.bus.BusBlock*.

Erbende Klassen

Von der Klasse erben folgende hier kurz mit Besonderheiten in der Funktionalität aufgelistete Klassen:

- *AluBlock*
- *BusBlock*
- *RegisterBlock*
- *SystemClockBlock*
- *ControlUnitBlock*
- *MemoryBlock*

ComputerBlockEntity<CL37>

Aufgabe

abstract BlockEntity für alle Computer Blöcke. Jeder *ComputerBlock* hat zur Laufzeit seine eigene eindeutige ComputerBlockEntity. *ComputerBlockEntity* erbt von *ModBlockEntity* und implementiert *IConnectableComputerBlockEntity* und *IGoggleQuerable*

Attribute

Die Klasse stellt folgende Attribute mit Sichtbarkeit *private* bereit:

Name	Typ	Einschränkungen	Beschreibung
model	IQueryable-BlockModel		Die Darstellung des Blocks im Modell, enthält die Daten des Blocks.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
tick	void	world: World, pos : BlockPos, state : BlockState, entity : ComputerBlockEntity	Methode, die von Minecraft bei jedem Tick aufgerufen wird. Ruft die Methode <i>ComputerBlockController::tick()</i> auf.

Name	Rückgabetyp	Parameter	Beschreibung
getComputer-Neighbours	List<Computer-BlockController>		Ermittelt die <i>ComputerBlockController</i> der benachbarten Blöcke dieses Blocks, die <i>BusBlock</i> sind. Es wird eine Liste aller Controller dieser zurückgegeben.
setBlockModel	void	model: IQueryablockModel	Legt das Modell für diesen Block fest. Der Parameter <i>model</i> enthält das Modell für diesen Block.
onBroken	void		Übergibt den onBroken-Aufruf an den <i>BlockController</i> , damit dieser ihn bearbeiten kann.
getGoggleText	Text		Bestimmt den Text, der angezeigt werden soll, wenn der Spieler auf diesen Block blickt und gibt ihn zurück.

Die Klasse stellt folgende Methoden mit Sichtbarkeit *protected* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
create-Controller	IUserInputReceivableComputerController		abstract Erstellt einen neuen, für den Block-Typ Spezifischen <i>Computer-Block-Controller</i> für diesen Block.

Zugehörigkeit

Die Klasse befindet sich im Paket *view.main.blocks.mod.computer* und nutzt die Klassen *controller.blocks.BlockController*, *controller.blocks.ComputerBlockController*, *controller.blocks.IConnectableComputerBlockEntity*, *controller.blocks.IUserInputReceivableComputerController*, *model.blocks.IQueryablockModel*, *model.data.IDataElement*, *view.main.blocks.mod.EntityType*, *view.main.blocks.mod.ModBlockEntity*, *view.main.blocks.mod.computer.bus.BusBlock*.

Erbende Klassen

Von der Klasse erben folgende hier kurz mit Besonderheiten in der Funktionalität aufgelistete Klassen:

- *BusBlockEntity* bietet zusätzlich die Methode mit Sichtbarkeit *public* an:

Name	Rückgabetyp	Parameter	Beschreibung
getComputer-Neighbours	List<Block-Controller>		Ermittelt die <i>BlockController</i> der Nachbar Blöcke dieses Blocks, die <i>Computer-Blocks</i> sind. Es wird eine Liste aller dieser Controller zurückgegeben,

- *AluBlockEntity*
- *RegisterBlockEntity*
- *SystemClockBlockEntity*
- *ComputerBlockEntityWithInventory <CL38>* bietet zusätzlich ein Inventar an.

ComputerBlockEntityWithInventory <CL38>

Aufgabe

abstract Diese Klasse bietet eine Standardimplementierung für eine *ComputerBlockEntity* mit einem Inventar. Erbt von *ComputerBlockEntity* und implementiert *ImplementedInventory*.

Attribute

Die Klasse stellt folgende Attribute mit Sichtbarkeit *protected* bereit:

Name	Typ	Einschränkungen	Beschreibung
items	DefaultedList<- ItemStack>		final Die Gegenstände im Inventar dieses Blocks.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
getItems	DefaultedList<- ItemStack>		Ermittelt die Gegenstände im Inventar dieses <i>Computer-Blocks</i> und gibt sie zurück.

Zugehörigkeit

Die Klasse befindet sich im Paket `view.main.blocks.mod.computer` und nutzt die Klasse `view.main.blocks.modImplementedInventory`.

Erbende Klassen

Von der Klasse erben folgende hier kurz mit Besonderheiten in der Funktionalität aufgelistete Klassen:

- *ControlUnitBlockEntity*

Kann in ein einelementiges Inventar ein *InstructionsetItem* aufnehmen, um die Architektur des Computers festzulegen.

- *MemoryBlockEntity*

Kann in ein einelementiges Inventar ein *ProgramItem* aufnehmen, um ein Programm in den Speicher zu laden.

Programming ProgrammingBlock(CL39)

Aufgabe

Diese Klasse definiert alle Programmierblöcke im Spiel. Sie erbt vom ModBlock.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetypr	Parameter	Beschreibung
createBlock-Entity	BlockEntity	pos: BlockPos, state : BlockState	Erzeugt eine neue Entität für einen Programmierblock. Diese Methode wird von Minecraft aufgerufen, wenn der Block geladen wird. <i>pos</i> Die Position des Blocks in der Minecraft-Welt, für den die Entität erstellt werden soll. <i>state</i> Der Zustand des Minecraft-Blocks, für den die Entität erstellt werden soll.

Zugehörigkeit

Die Klasse befindet sich im Paket `view.main.blocks.mod.programming` und nutzt die Klassen `view.main.blocks.mod.ModBlock`, `view.main.blocks.mod.computer.systemclock.SystemClockBlockEntity`.

ProgrammingBlockEntity<CL40>

Aufgabe

Diese Klasse repräsentiert eine Programmierblock-Entität aus unserem Mod im Spiel. Jeder Programmierblock hat seine eigene einzigartige ProgrammingBlockEntity, während er geladen wird. Erbt von `ModBlockEntityWithinInventory`, implementiert `ExtendedScreenHandlerFactory`

Attribute

Die Klasse stellt folgende Attribute mit Sichtbarkeit `private` bereit:

Name	Typ	Einschränkungen	Beschreibung
code	String		Der Code, der sich derzeit im Programmierblock befindet. Er ist möglicherweise nicht auf dem neuesten Stand mit dem Code im Programmierbildschirm des Clients.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit `public` bereit:

Name	Rückgabetyp	Parameter	Beschreibung
writeScreenOpeningData	void	player: ServerPlayerEntity, buf : PacketByteBuf	Schreibt die zum Öffnen des Bildschirms benötigten Daten in den <code>PacketByteBuf</code> . Der Parameter <code>player</code> enthält den Spieler, der den Bildschirm öffnet. Der Parameter <code>buf</code> enthält den <code>PacketByteBuf</code> , in den geschrieben werden soll.

Name	Rückgabetyp	Parameter	Beschreibung
getDisplay-Name	minecraft.Text		Gibt den Anzeigenamen des Bildschirms zurück
createMenu	ScreenHandler	syncId: int, playerInventory : PlayerInventory , player : Player-Entity	Erzeugt den <i>ProgrammingScreenHandler</i> . Wird von Minecraft aufgerufen.
setCode	void	code: String	Setzt den Code des Programmierblocks. Der Parameter <i>code</i> enthält den zu setzenden Code.
getCode	String		Gibt den Code des Programmierblocks zurück.

Die Klasse stellt folgende Methoden mit Sichtbarkeit *protected* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
create-Controller	IUserInput- Receivable- Controller		Der Rückgabewert ist ein <i>ProgrammingController</i> gebunden an dieses <i>BlockEntity</i> .

Zugehörigkeit

Die Klasse befindet sich im Paket *view.main.blocks.mod.programming* und nutzt die Klassen *controller.assembler.AssemblyException*, *controller.blocks.IUserInputReceivableController*, *controller.blocks.ProgrammingController*, *model.data.IDataElement*, *view.main.RISCJ_blockits*, *view.main.blocks.modImplementedInventory*, *view.main.blocks.mod.ModBlockEntityWithInventory*, *view.main.data.DataNbtConverter*, *view.main.data.NbtDataConverter*.

ProgrammingScreenHandler⟨CL41⟩

Aufgabe

Diese Klasse bietet Funktionen zur Synchronisierung des Programmierbildschirms zwischen Client und Server. Sie fügt alle Slots und Schaltflächen auf dem Bildschirm hinzu. Bietet Funktionalität für die Behandlung von Schaltflächenklicks auf dem Bildschirm. Sie erbt von *ModScreenHandler*.

Attribute

Die Klasse stellt folgende Attribute mit Sichtbarkeit *private* bereit:

Name	Typ	Einschränkungen	Beschreibung
ASSEMBLE_- BUTTON_ID	int		static final Die ID der Schaltfläche "assemble".
blockEntity	Programming- BlockEntity		Die <i>BlockEntity</i> , die diesen <i>ScreenHandler</i> erstellt hat.
inventory	Inventory		Das Inventar des Programmierblocks.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
onButtonClick	boolean	player: Player- Entity, id : int	Verarbeitet Mausklicks auf den Knöpfen des Bildschirms. Ein Klick auf die Schaltfläche "assemble" löst das assembeln im Programmierblock aus. Der Parameter <i>player</i> enthält den Spieler, der die Schaltfläche angeklickt hat. Der Parameter <i>id</i> enthält die ID der Schaltfläche. Die Methode gibt true zurück, wenn die Schaltfläche angeklickt wurde, andernfalls false.

Die Klasse stellt folgende Methoden mit Sichtbarkeit *private* bereit¹¹:

Name	Rückgabetyp	Parameter	Beschreibung
showError	void	message: String	Zeigt eine Fehlermeldung auf dem Bildschirm an. Der Parameter <i>message</i> enthält die Meldung, die angezeigt werden soll.
addProgram-Slot	void		Fügt den Slot für das <i>ProgramItem</i> auf dem Bildschirm hinzu.
add-Instruction-SetSlot	void		Fügt den Slot für das <i>InstructionSetItem</i> auf dem Bildschirm hinzu.

Zugehörigkeit

Die Klasse befindet sich im Paket *view.main.blocks.mod.programming* und nutzt die Klassen *controller.assembler.AssemblyException*, *view.main.RISCJ_blockits*, *view.main.blocks.mod.-ModScreenHandler*, *view.main.items.instructionset.InstructionsetItem*.

Data Converter NbtDataConverter(CL42)

Aufgabe

Wandelt *NbtCompound* in *IDataElement* um. Dies geschieht durch rekursiven Besuch der *NbtElements*. Implementiert *NbtElementVisitor*.

Attribute

Die Klasse stellt folgende Attribute mit Sichtbarkeit *private* bereit:

Name	Typ	Einschränkungen	Beschreibung
data	IDataElement		Das Datenelement.

¹¹ggfs. werden private Hilfsmethoden nicht dargestellt

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
NbtData-Converter	NbtData-Converter	compound : Nbt-Compound	Konstructor, initialisiert <i>data</i> und besucht den <i>NbtCompound</i> .
visitString	void	element: Nbt-String	Besucht einen <i>NbtString</i> und erstellt einen <i>DataStringEntry</i> . Der Parameter <i>element</i> enthält den zu besuchenden <i>NbtString</i> .
visitByte	void	element: NbtByte	Besucht ein <i>NbtByte</i> . Der Parameter <i>element</i> enthält das zu besuchende <i>NbtByte</i> .
visitShort	void	element: Nbt-Short	Besucht ein <i>NbtShort</i> . Der Parameter <i>element</i> enthält das zu besuchende <i>NbtShort</i> .
visitInt	void	element: NbtInt	Besucht ein <i>NbtInt</i> . Der Parameter <i>element</i> enthält das zu besuchende <i>NbtInt</i> .
visitLong	void	element: NbtLong	Besucht eine <i>NbtLong</i> . Der Parameter <i>element</i> enthält die zu besuchende <i>NbtLong</i> .
visitFloat	void	element: Nbt-Float	Besucht ein <i>NbtFloat</i> . Der Parameter <i>element</i> enthält die zu besuchende <i>NbtFloat</i> .
visitDouble	void	element: Nbt-Double	Besucht ein <i>NbtDouble</i> . Der Parameter <i>element</i> enthält das zu besuchende <i>NbtDouble</i> .
visitByteArray	void	element: Nbt-ByteArray	Besucht ein <i>NbtByteArray</i> . Der Parameter <i>element</i> enthält das zu besuchende <i>NbtByteArray</i> .

Name	Rückgabetyp	Parameter	Beschreibung
visitIntArray	void	element: NbtIntArray	Besucht ein <i>NbtIntArray</i> . Der Parameter <i>element</i> enthält das zu besuchende <i>NbtIntArray</i> .
visitLong-Array	void	element: NbtLongArray	Besucht ein <i>NbtLongArray</i> . Der Parameter <i>element</i> enthält das zu besuchende <i>NbtLongArray</i> .
visitList	void	element: NbtList	Besucht eine <i>NbtList</i> . Der Parameter <i>element</i> enthält die zu besuchende <i>NbtList</i> .
visit-Compound	void	compound: NbtCompound	Besucht einen <i>NbtCompound</i> . Erzeugt eine neue Data und besucht alle Schlüssel. Der Parameter <i>compound</i> enthält den zu besuchenden <i>NbtCompound</i> .
visitEnd	void	element: NbtEnd	Besucht ein <i>NbtEnd</i> . Der Parameter <i>element</i> enthält das zu besuchende <i>NbtEnd</i> .
getData	IDataElement		Gibt das konvertierte <i>IDataElement</i> zurück. Der Rückgabewert ist das konvertierte <i>IDataElement</i> .

Zugehörigkeit

Die Klasse befindet sich im Paket *view.main.data* und nutzt die Klassen *model.data.Data*, *model.data.DataStringEntry*, *model.data.IDataContainer*, *model.data.IDataElement*.

DataNbtConverter $\langle CL43 \rangle$

Aufgabe

Konvertiert *IDataElement* in *NbtCompound*. Dies geschieht durch rekursiven Besuch der *IDataElemente*. Implementiert *IDataVisitor*.

Attribute

Die Klasse stellt folgende Attribute mit Sichtbarkeit *private* bereit:

Name	Typ	Einschränkungen	Beschreibung
compound	NbtElement		Der NbtCompound.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
DataNbt- Converter	DataNbt- Converter	element : IData- Element	Konstructor, initialisiert den <i>NbtCompound</i> und besucht das <i>IDataElement</i> .
visit	void	container: IData- Container	Besucht einen <i>IDataContainer</i> und besucht rekursiv dessen Inhalte. Der Parameter <i>container</i> enthält den zu besuchenden <i>IDataContainer</i> .
visit	void	element: IData- StringEntry	Besucht einen <i>IDataStringEntry</i> . Der Parameter <i>element</i> enthält den zu besuchenden <i>DataStringEntry</i> .
getNbt- Element	NbtElement		Ruft das <i>NbtElement</i> ab. Der Rückgabewert ist das <i>NbtElement</i> .

Zugehörigkeit

Die Klasse befindet sich im Paket *view.main.data* und nutzt die Klassen *model.data.IDataContainer*, *model.data.IDataElement*, *model.data.IDataStringEntry*, *model.data.IDataVisitor*.

Items GogglesItem $\langle CL44 \rangle$

Aufgabe

Diese Klasse definiert den Gegenstand "Brille" im Spiel. Implementiert *Equipment*, erbt von *Item*.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
getSlotType	EquipmentSlot		Gibt den Slot-Typ des Gegenstands Goggles zurück. Damit wird Minecraft mitgeteilt, wo das Item ausgerüstet werden kann. Der Rückgabewert return Der Slot-Typ des Items goggles.

Zugehörigkeit

Die Klasse befindet sich im Paket *view.main.items.goggles*.

InstructionsetItem $\langle CL45 \rangle$

Aufgabe

Diese Klasse definiert den Befehlssatz im Spiel. Erbt von *Item*.

Attribute

Die Klasse stellt folgende Attribute mit Sichtbarkeit *private* bereit:

Name	Typ	Einschränkungen	Beschreibung
default-Instruction-SetJson	String		Der Standard-Befehlssatz json.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
inventoryTick	void	stack: ItemStack, world : World, en- tity: Entity, slot : int, selected : bo- lean	Wird jeden Minecraft-Tick aufgerufen. Setzt, solange keine anderer vorhanden, den Standardbefehlssatz json auf den <i>ItemStack</i> . Der Parameter <i>stack</i> enthält den <i>ItemStack</i> . Der Parameter <i>world</i> enthält die Welt. Der Parameter <i>entity</i> enthält das <i>Entity</i> , welches das Item hält. Der Parameter <i>slot</i> enthält den index des slots in dem sich der <i>ItemStack</i> befindet. Der Parameter <i>selected</i> be- obreibt ob der <i>ItemStack</i> ausgewählt ist

Zugehörigkeit

Die Klasse befindet sich im Paket *view.main.items.instructionset*.

ManualItem(CL46)**Aufgabe**

Diese Klasse definiert das Handbuch im Spiel. Erbt von *Item*.

Operationen

Die Klasse stellt folgende Methoden mit Sichtbarkeit *public* bereit:

Name	Rückgabetyp	Parameter	Beschreibung
use	TypedAction-Result-<ItemStack>	world : World, user: Player-Entity, hand : Hand	Wird beim Rechtsklicken mit diesem Item in der Hand aufgerufen. Öffnet einen Bildschirm, der eine Anleitung zur Modifikation gibt.

Zugehörigkeit

Die Klasse befindet sich im Paket *view.main.items.manual*.

2.2 Klassendiagramme

2.2.1 View

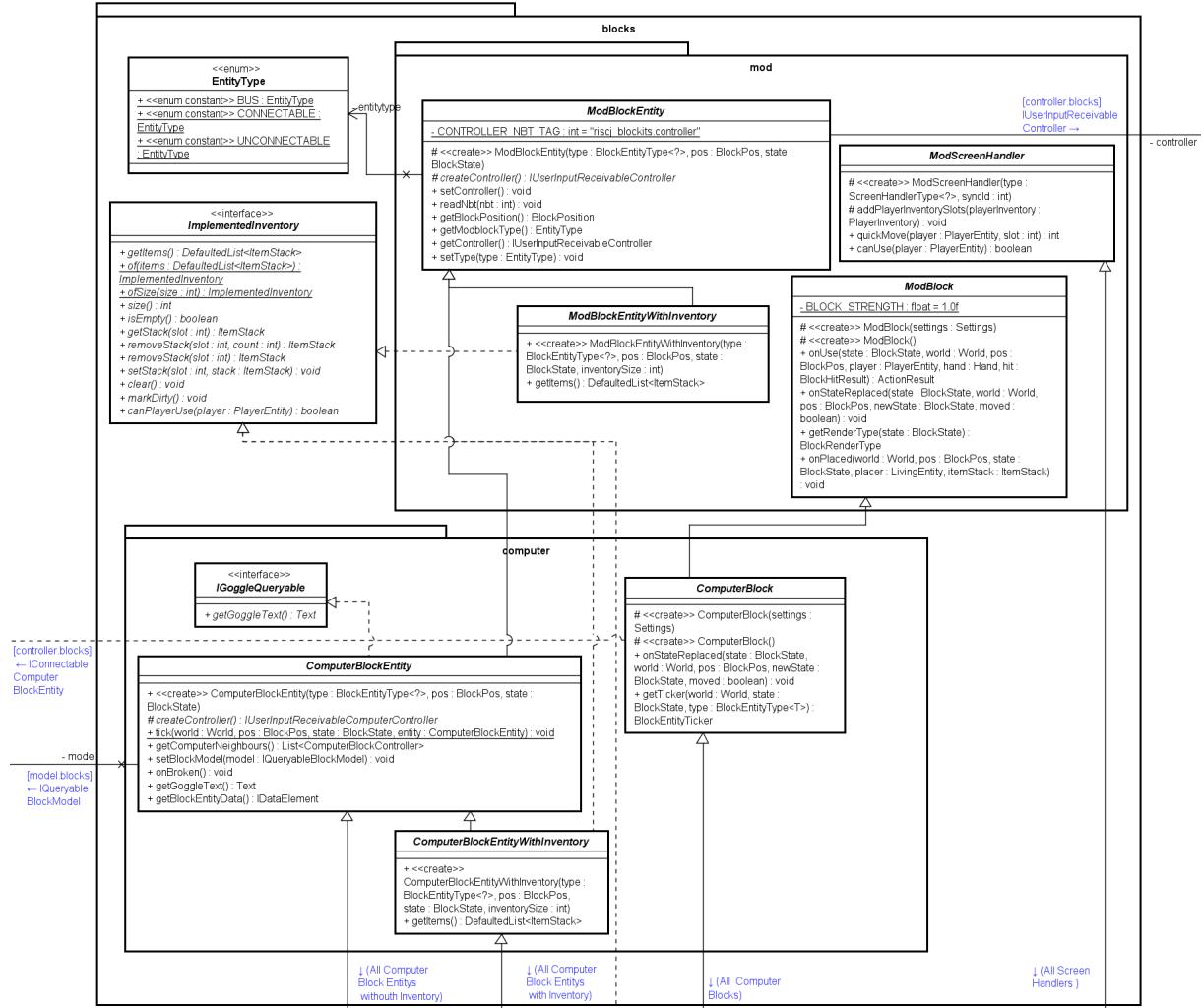


Abbildung 2.2: 1. Klassendiagramm View

Abbildung 2.2 zeigt die abstrakte Hierarchieebene der Fabric-Main-Komponente. Hier werden durch abstrakte Klassen Gemeinsamkeiten in den *Block*- und *BlockEntity*-Klassen festgelegt. Ein *ComputerBlock* <CL36> ist ein Block, der Teil eines Computers sein kann.

PRAXIS DER SOFTWAREENTWICKLUNG

RISCJ Blockits

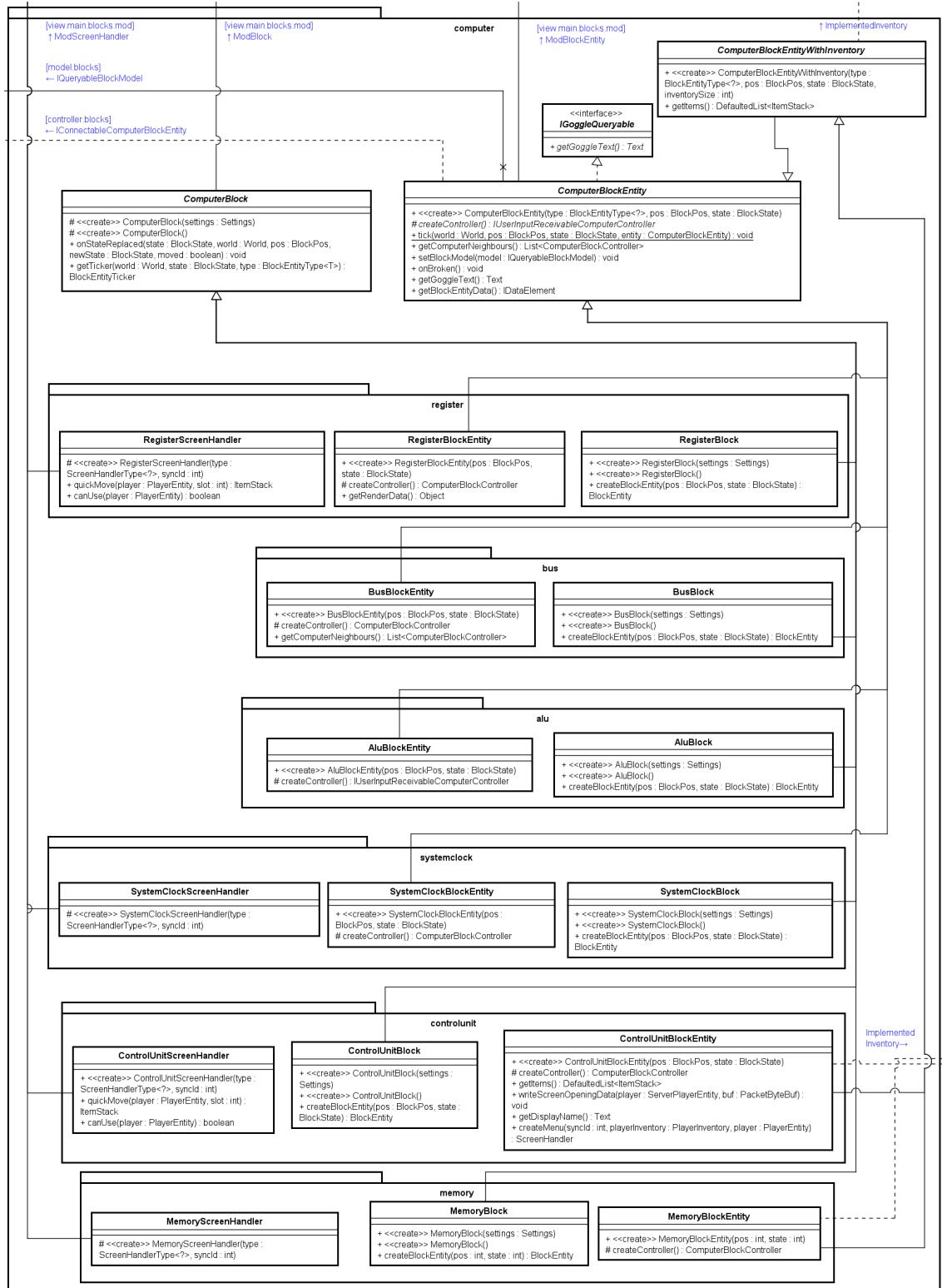


Abbildung 2.3: 2. Klassendiagramm View

Abbildung 2.3 zeigt den abstrakten `ComputerBlock` $\langle CL36 \rangle$ und die davon erbenden speziellen Blöcke.

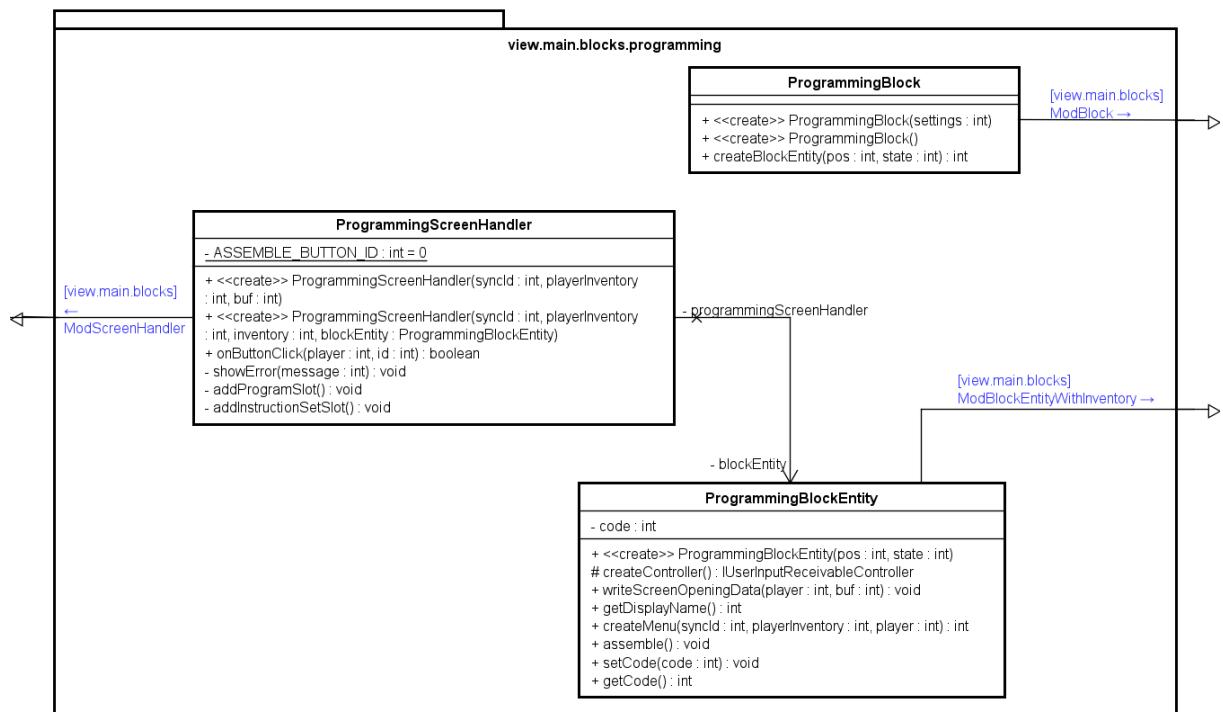


Abbildung 2.4: Klassendiagramm Programming im View

Abbildung 2.4 zeigt den *ProgrammingBlock* $\langle CL39 \rangle$, der nicht Teil eines Computers ist und nur vom *ModBlock* $\langle CL32 \rangle$ erbt.

PRAXIS DER SOFTWAREENTWICKLUNG

RISCJ Blockits

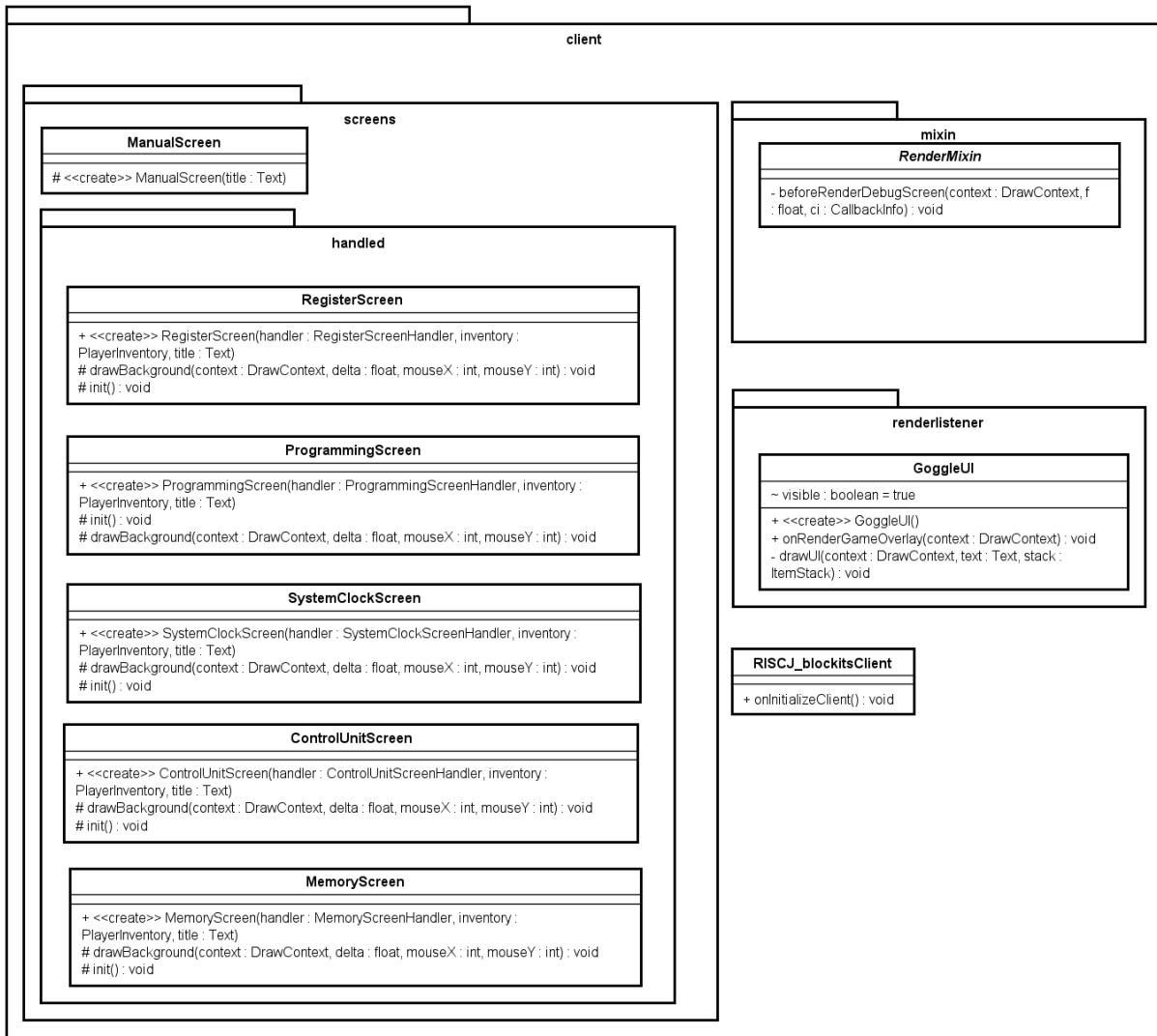


Abbildung 2.5: Klassendiagramm Client

Abbildung 2.5 zeigt alle Klassen in der Fabric-Client-Komponente. Sie wird anders als alle anderen Komponenten nur im (logischen) Client von Minecraft ausgeführt und ist damit zuständig für das Rendern von User Interfaces.

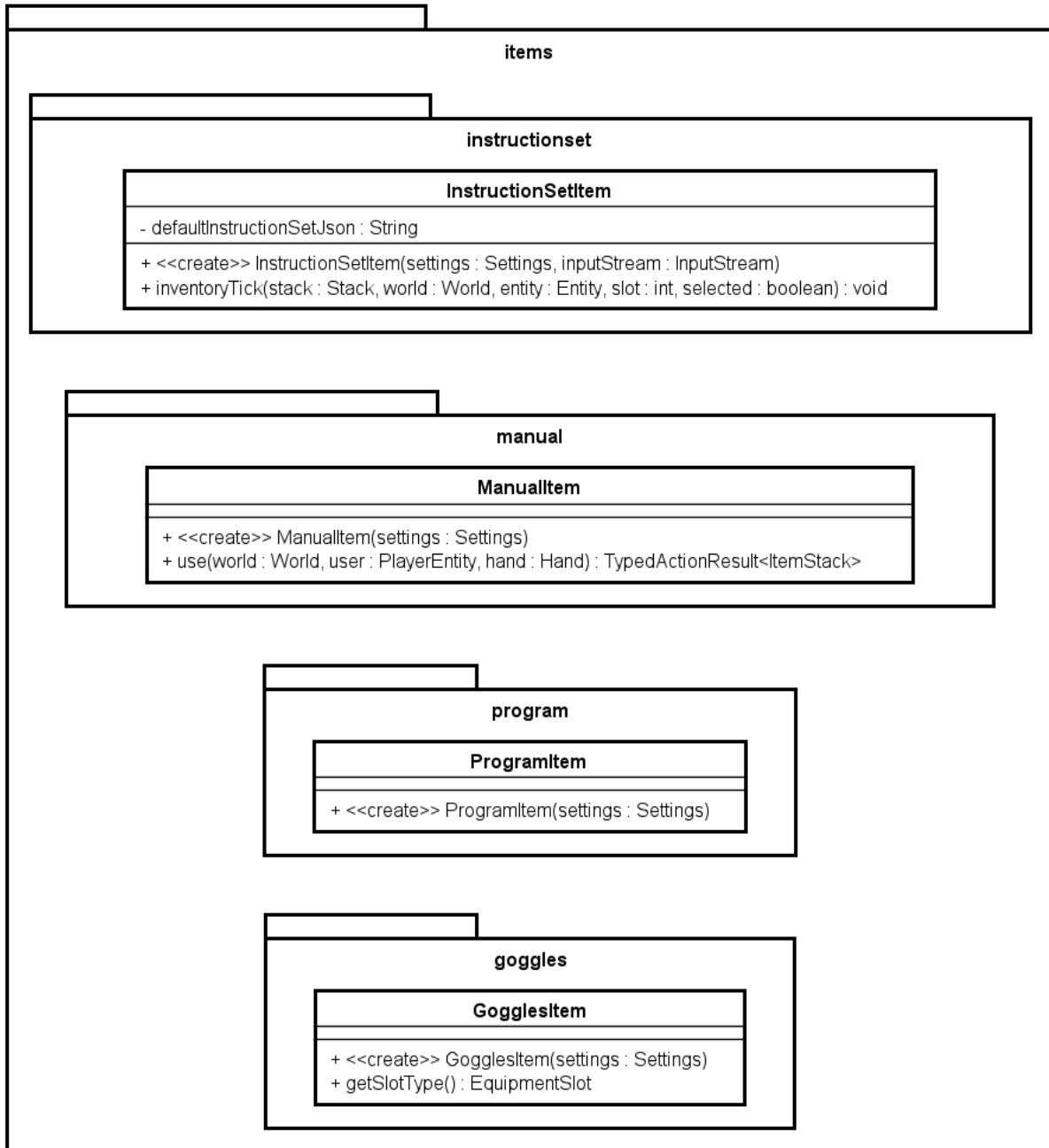


Abbildung 2.6: Klassendiagramm Items

Abbildung 2.6 zeigt die Klassen, die Items im Spiel definieren. Diese bieten Methoden, die dann für die jeweiligen *ItemStacks* ausgeführt werden.

2.2.2 Controller

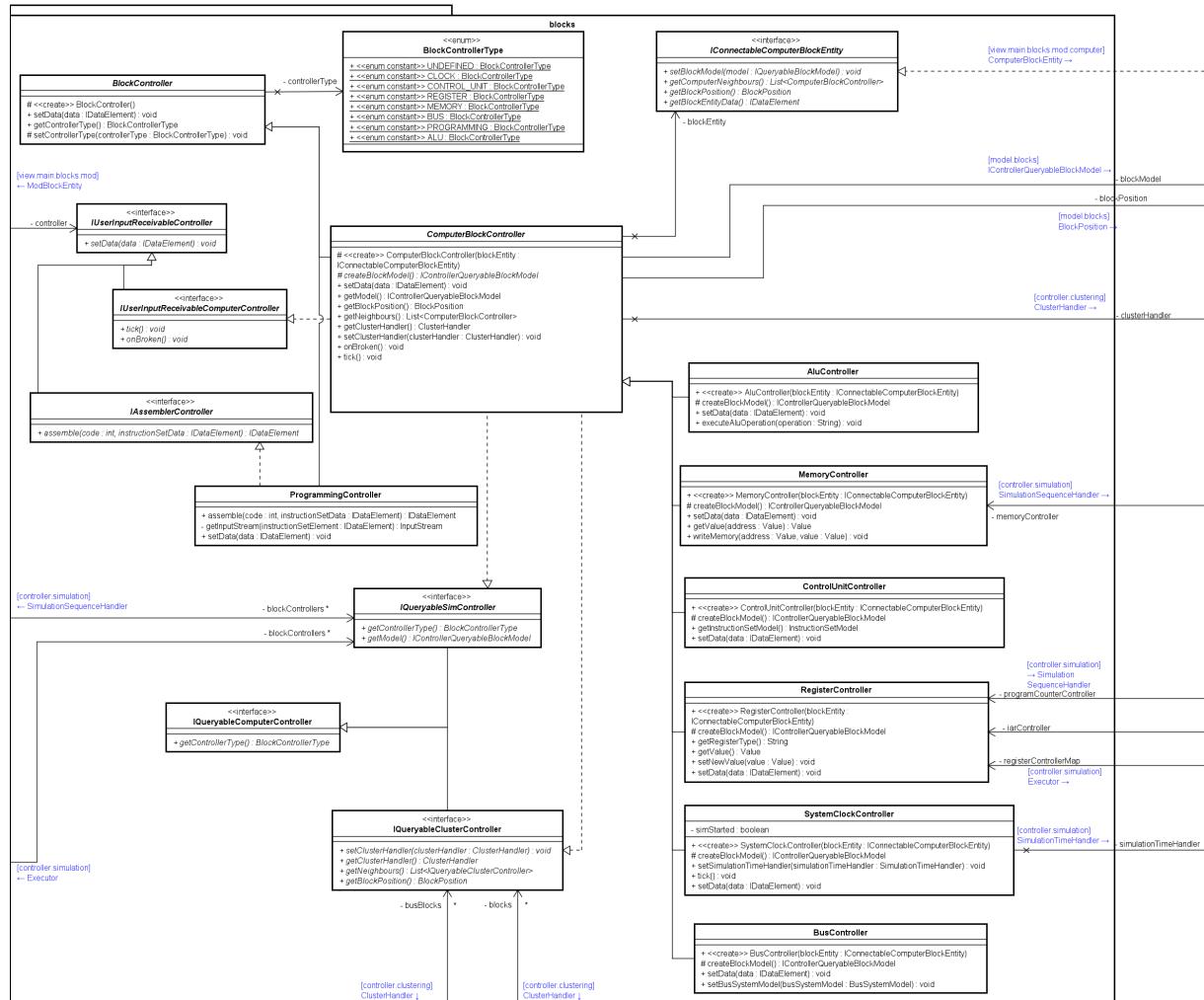


Abbildung 2.7: Klassendiagramm Block-Controller

Abbildung 2.7 zeigt Block-Controller. Sie nehmen eine zentrale Rolle ein, da sie alle Benutzereingaben aus dem View verarbeiten, und dann auf dem Model die Simulation ausführen. Auch hier gibt es wie bei den *ModBlockEntitys* *CL33* *BlockController* *CL4* die nicht zum Computer gehören und deswegen kein *BlockModel* *CL11* haben und die spezialisierten *ComputerBlockController* *CL5*. Ein *BlockController* *CL4* ist immer einer *ModBlockEntity* *CL33* zugeordnet.

PRAXIS DER SOFTWAREENTWICKLUNG

RISCJ Blockits

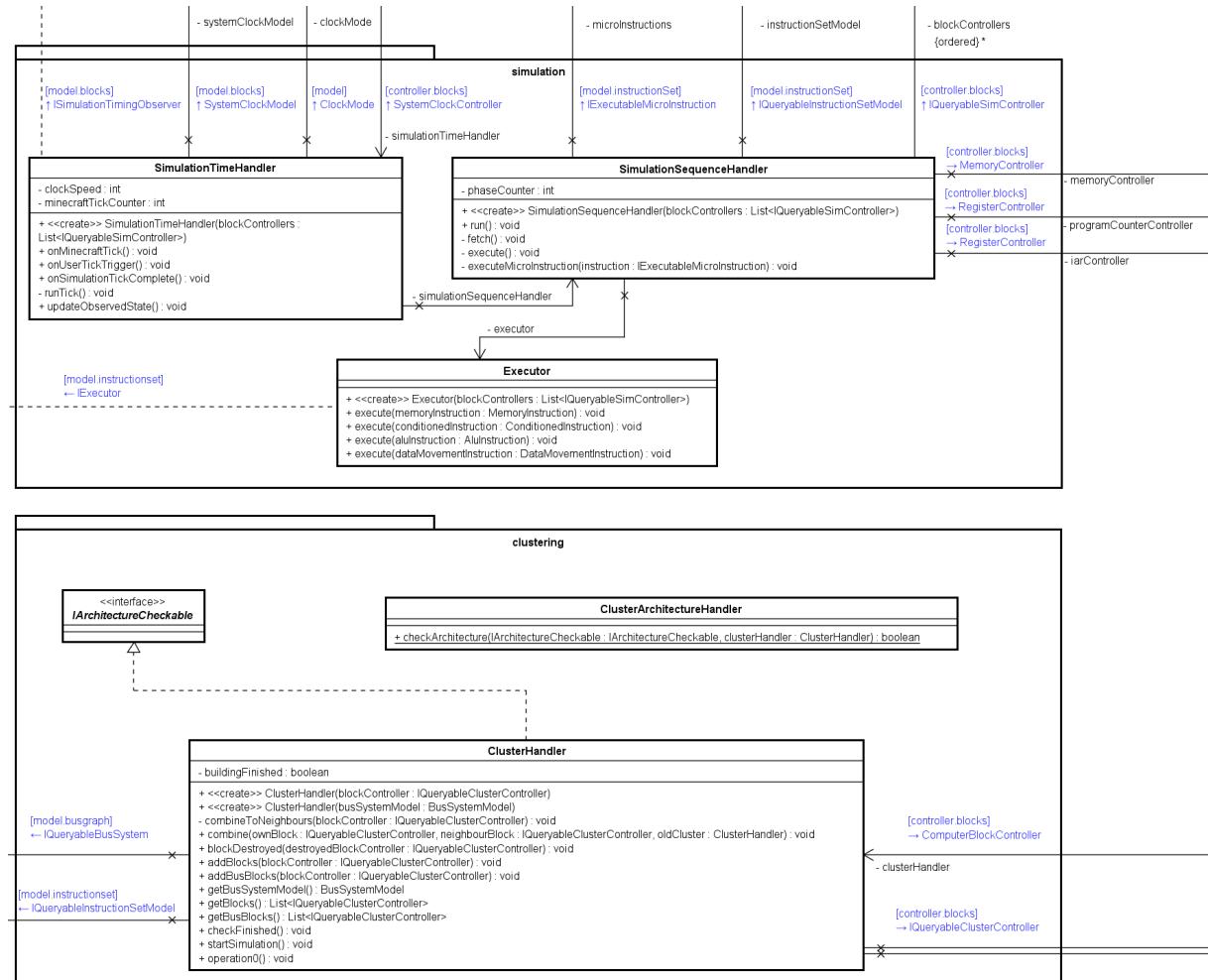


Abbildung 2.8: Klassendiagramm Simulation und Clustering

Abbildung 2.8 zeigt die übergeordneten Controller. Anders als die *BlockController* (*CL4*) verwalteten sie keine einzelnen Blöcke, sondern eine Gruppe von Block-Controllern. Der *ClusterHandler* (*CL6*) ist in der Aufbauphase aktiv, während der Benutzer den Computer aufbaut. Er sammelt Blöcke, die mit dem gleichen Bus verbunden sind und erstellt, wenn der Computer komplett ist, die Simulation. Der *SimulationTimeHandler* (*CL9*) verwaltet, wann die einzelne Simulationschritte ausgeführt werden. Entweder in Abhängigkeit von Minecraft-Ticks oder wenn der vorhergegangene fertig ist. Der *SimulationSequenceHandler* (*CL8*) verwaltet dann den einzelnen Tick. Je nach Ausführungsphase und Befehlssatz müssen verschiedene Dinge passieren. Für die Ausführung der Operationen wird der *Executor* (*CL7*) benutzt, der *MicroInstruktionen* (*CL20*) auf den *ComputerBlockControllern* (*CL5*) ausführt, die dann wiederum die auf ihnen ausgeführten Operationen auf dem jeweiligen *BlockModel* (*CL11*) ausführen.

PRAXIS DER SOFTWAREENTWICKLUNG

RISCJ Blockits

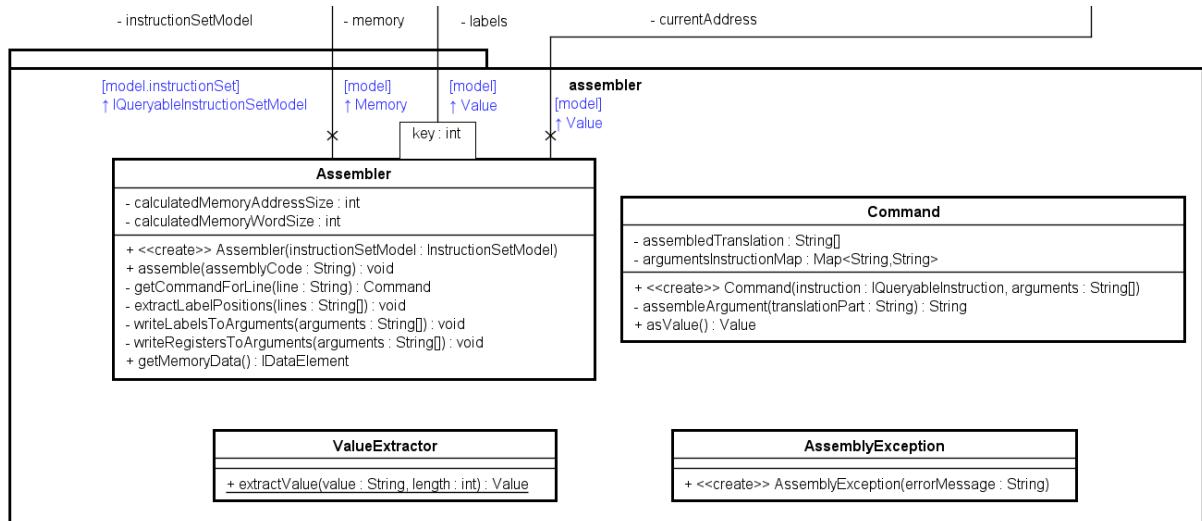


Abbildung 2.9: Klassendiagramm Assembler

Abbildung 2.9 zeigt den *Assembler*. Er wird vom *ProgrammingController* genutzt, um vom Benutzer in Assembler-Sprache eingegebene Programme in Maschinencode zu übersetzen. Dabei sind die Einzelheiten beider Sprachen und die Übersetzungsweise im Befehlssatz definiert.

2.2.3 Model

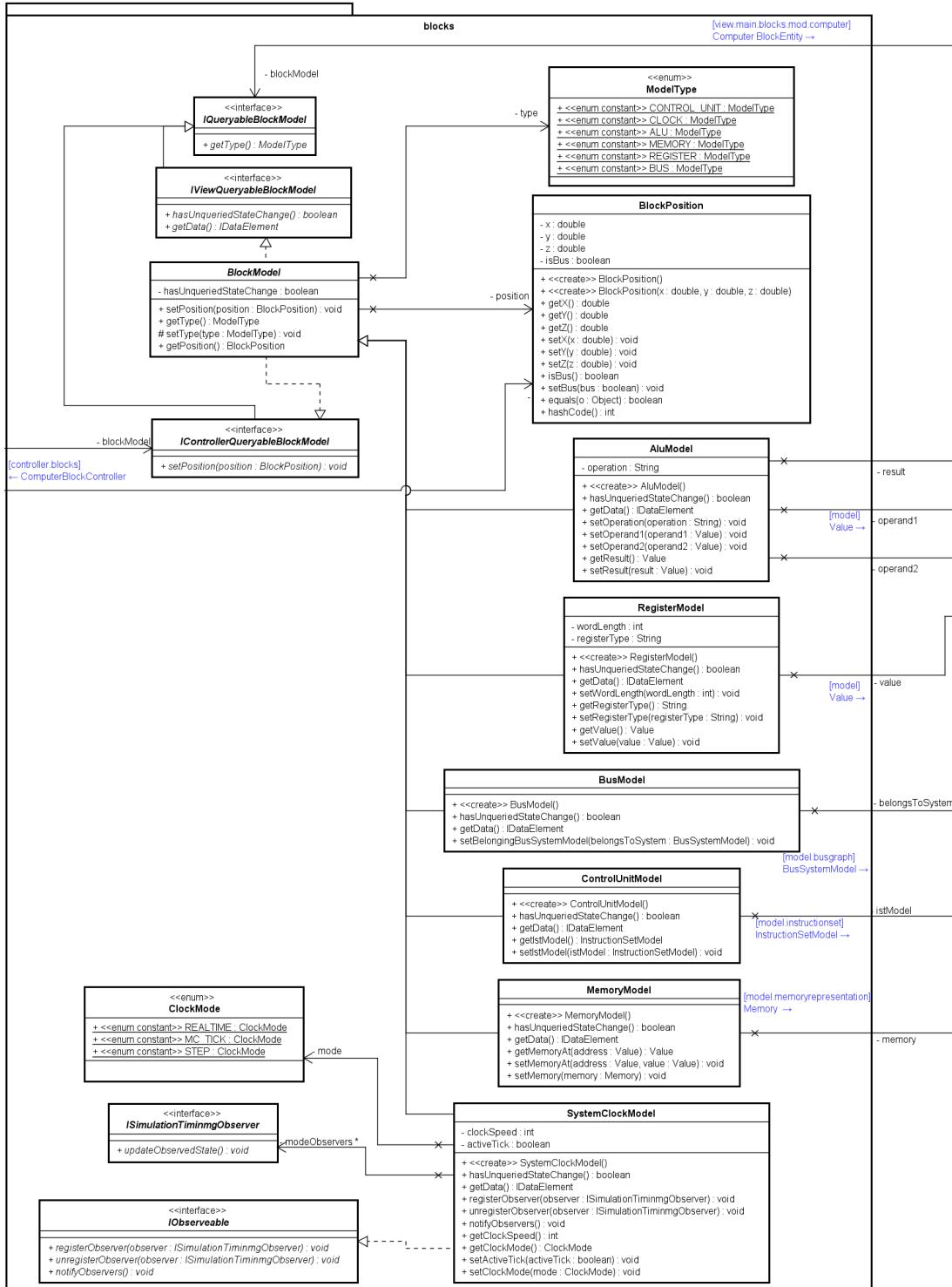


Abbildung 2.10: Klassendiagramm Block-Model

Abbildung 2.10 zeigt die Hardware-Elements-Komponente im Model. In ihr haben alle *ModBlockEntitys* $\langle CL33 \rangle$ eine Repräsentation, die ihren Status speichert. Er kann von den dazugehörigen *BlockControllern* $\langle CL4 \rangle$ geändert werden und von den *ModBlockEntitys* $\langle CL33 \rangle$ auf Veränderungen abgefragt werden. Da Minecraft mit Ticks arbeitet, ergibt es Sinn diese Abfrage jeden Tick zu machen. Nur dann kann sich der Block auch ändern. Minecraft hat ein eigenes System, um Daten persistent zu speichern. Deswegen können wir das Model nicht speichern, sondern nur die Daten, die in ihm liegen. Solange ein Spieler in der Nähe des Blocks ist, existiert auch die *ModBlockEntity* $\langle CL33 \rangle$ und damit unser Model. Wird der Block entladen, speichern wir die Daten des *BlockModels* $\langle CL11 \rangle$ im Minecraft-NBT-Format. Wird der Block wieder geladen, erstellen wir ein neues *BlockModel* $\langle CL11 \rangle$ mit den Daten aus dem NBT.

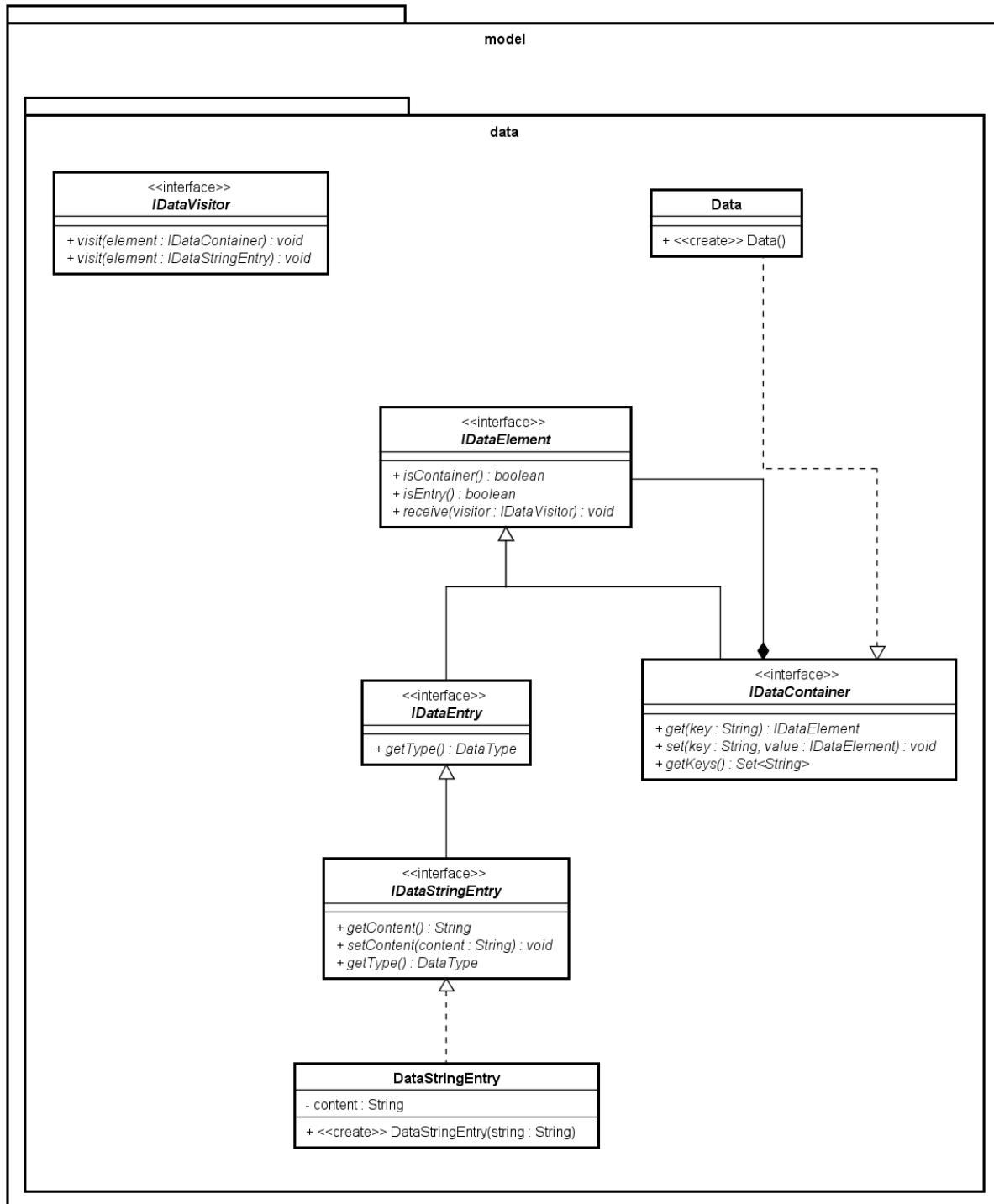


Abbildung 2.11: Klassendiagramm Data

Abbildung 2.11 zeigt, wie Daten in einer flexiblen Form aufgebaut sind, und dementsprechend einfach abgerufen werden können.

PRAXIS DER SOFTWAREENTWICKLUNG

RISCJ Blockits

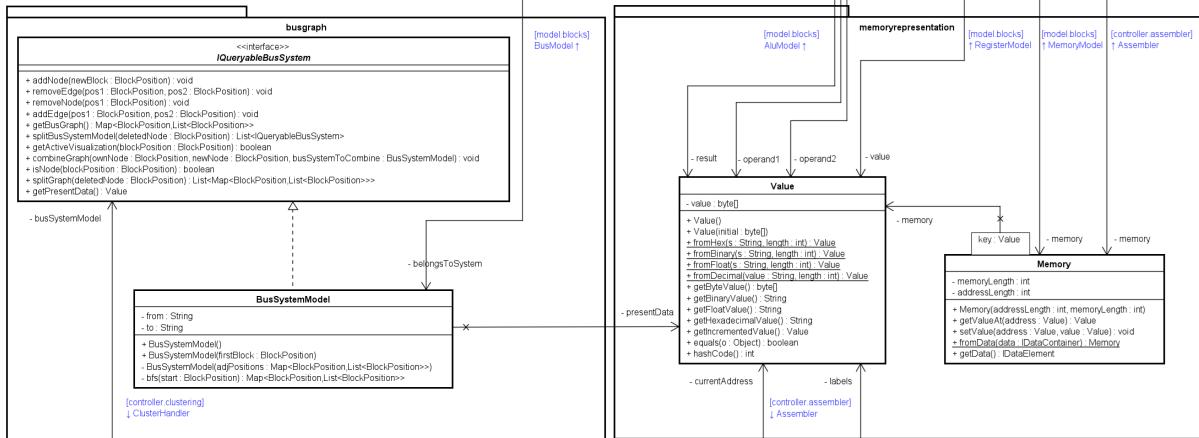


Abbildung 2.12: Klassendiagramm BusGraph und Memory

Abbildung 2.12 zeigt zwei wichtige Komponenten. Das *BusSystemModel* ⟨CL28⟩ speichert eine Graphen-Präsentation eines Computers und stellt Operationen auf dem Graphen bereit. Die *memoryrepresentation* stellt Präsentationen für einen Wert und einen Speicher, der aus vielen Werten mit Werten als Adressen besteht. Die Werte sind variabel festlegbar in der Größe.

PRAXIS DER SOFTWAREENTWICKLUNG

RISCJ Blockits

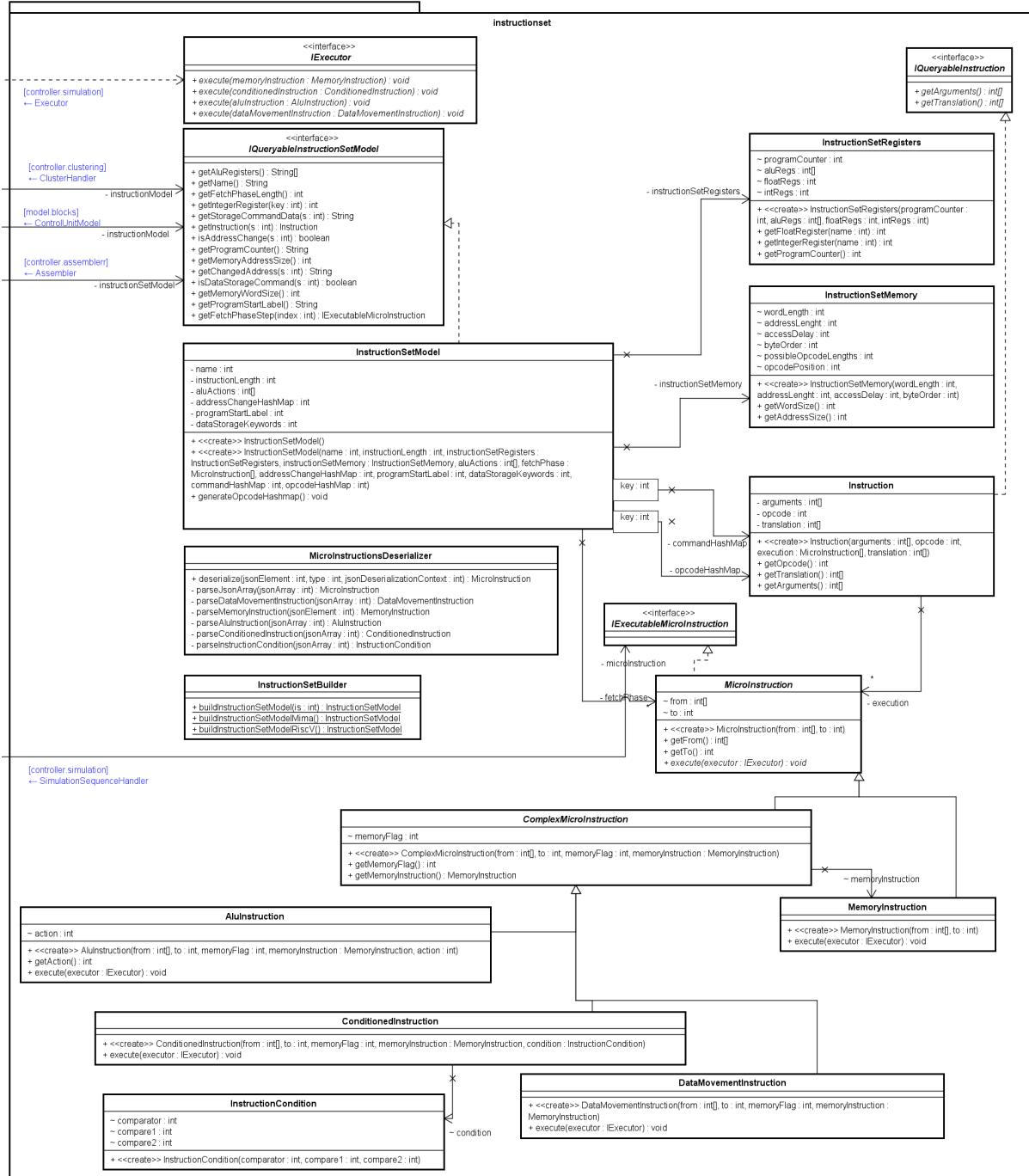


Abbildung 2.13: Klassendiagramm Instruction Set

Abbildung 2.13 zeigt die Instruction-Set-Komponente, die dynamisch aus einer .json-Datei geladen wird und definiert, wie sich der Computer verhält. Die *MicroInstructions* definieren einzelne Prozessortakte. Eine *Instruction* definiert einen Assembler-Befehl und enthält mehrere *MicroInstructions*.

2.3 Patterns

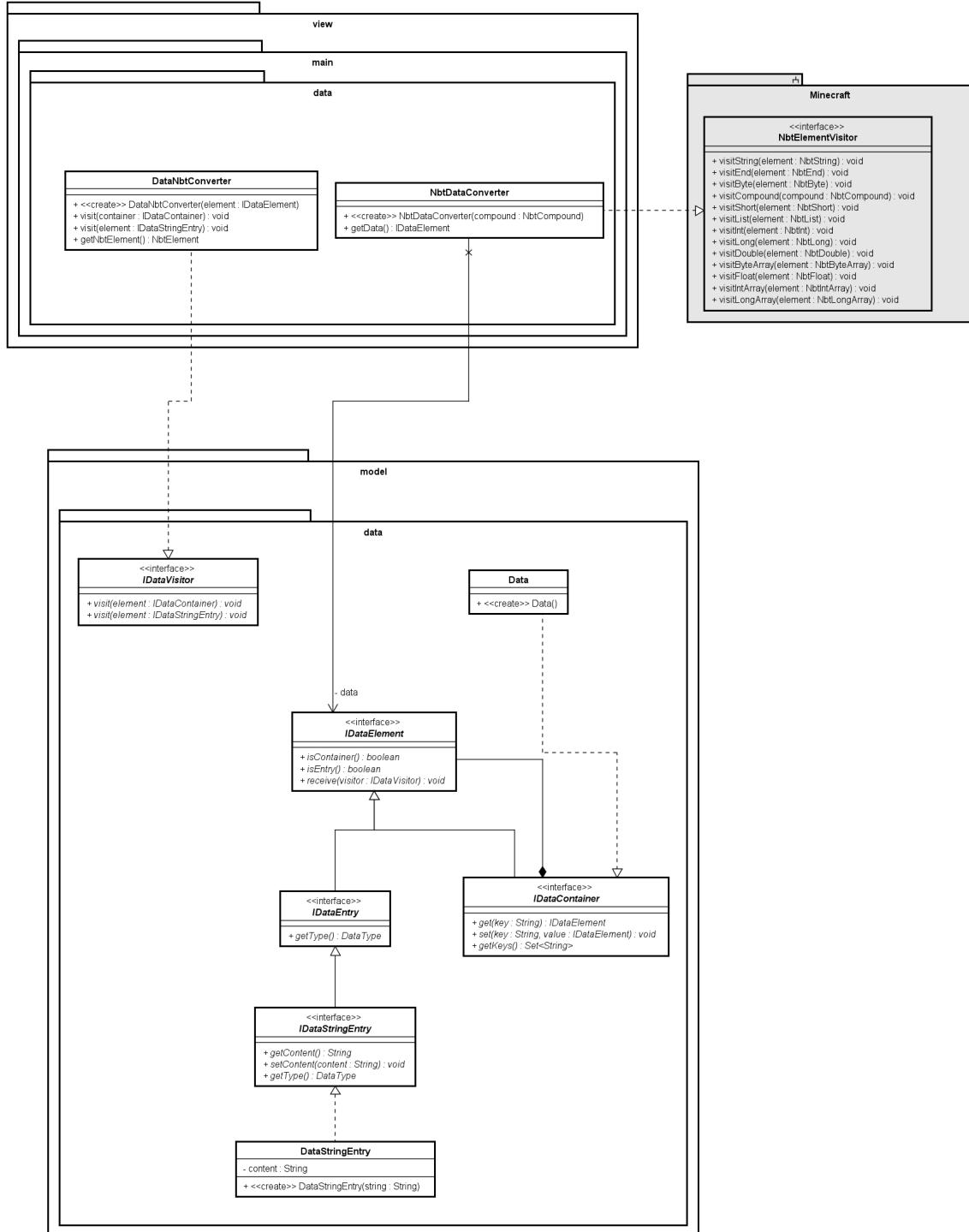


Abbildung 2.14: Klassendiagramm Data Composite-Visitor-Pattern

Abbildung 2.14 zeigt die interne Datenstruktur. Diese ist nach dem Composite-Pattern aufgebaut. Entsprechend kann ein *IDataContainer*, welcher selbst ein *IDataElement* ist, mehrere *IDataElemente* enthalten. Ein *IDataElement* kann auch ein *IDataEntry* sein, hier dann ein *IDataStringEntry*, in welchem ein Text steht.

Alle *IDataElemente* sind wiederum Teil eines Visitor-Patterns. Dies erlaubt es einem *IDataVisitor*, dieses Composite zu besuchen.

Genutzt werden dieser *IDataVisitor* und auch der von Minecraft gebotene *NbtElementVisitor*, um die beiden Formate IData und Nbt ineinander zu konvertieren.

PRAXIS DER SOFTWAREENTWICKLUNG

RISCJ Blockits

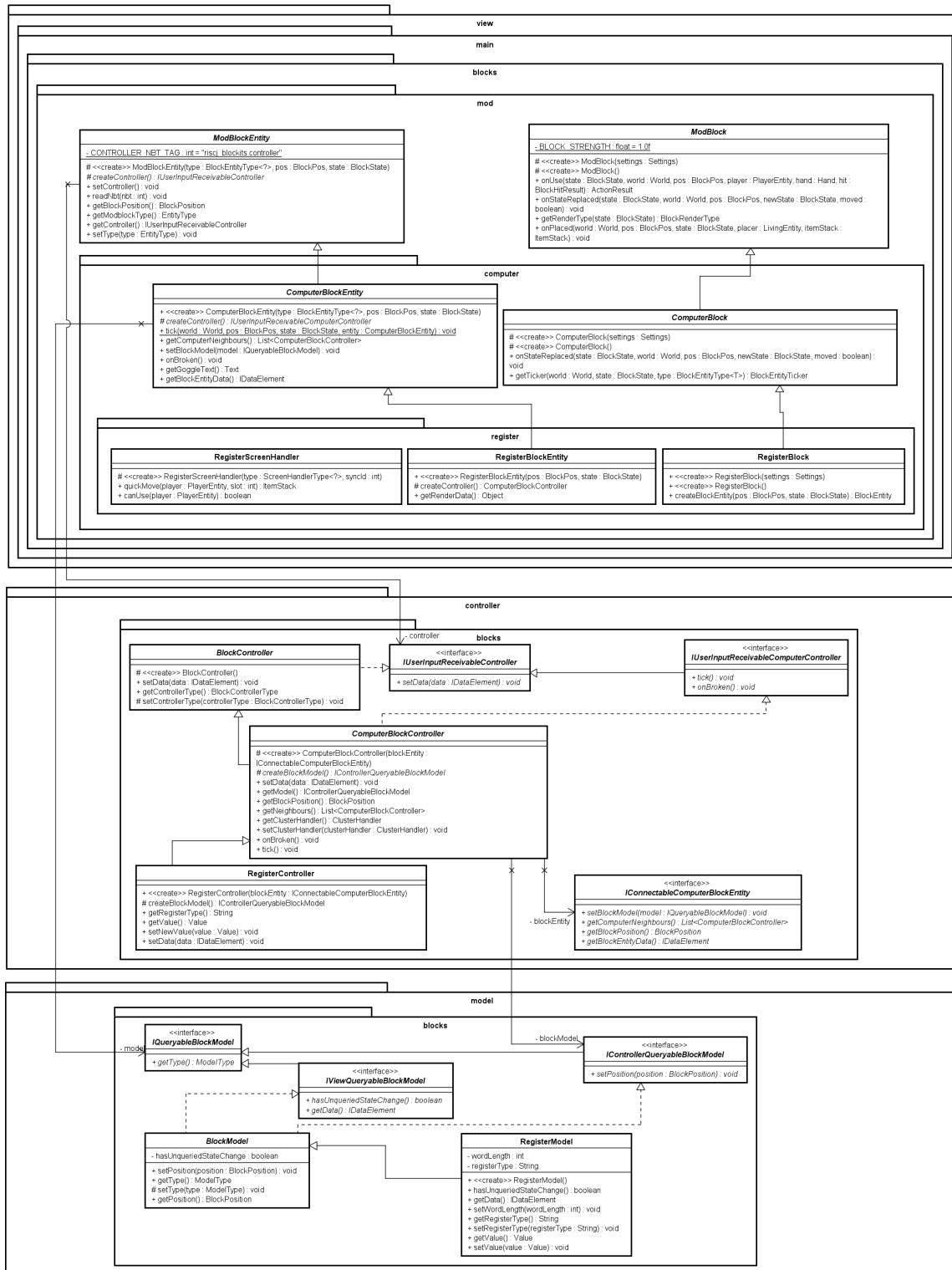


Abbildung 2.15: Klassendiagramm der Model-View-Controller-Struktur für die einzelnen Blöcke am Beispiel des Register-Blocks

Abbildung 2.15 stellt das Model-View-Controller-Architekturmuster für einen einzelnen Block dar. Eingebettet in das Model-View-Controller-Muster des Gesamtprojekts finden sich für jeden Block ein Model, View-Elemente entsprechend der Minecraft-Struktur, sowie ein Controller, welche über Interfaces kommunizieren.

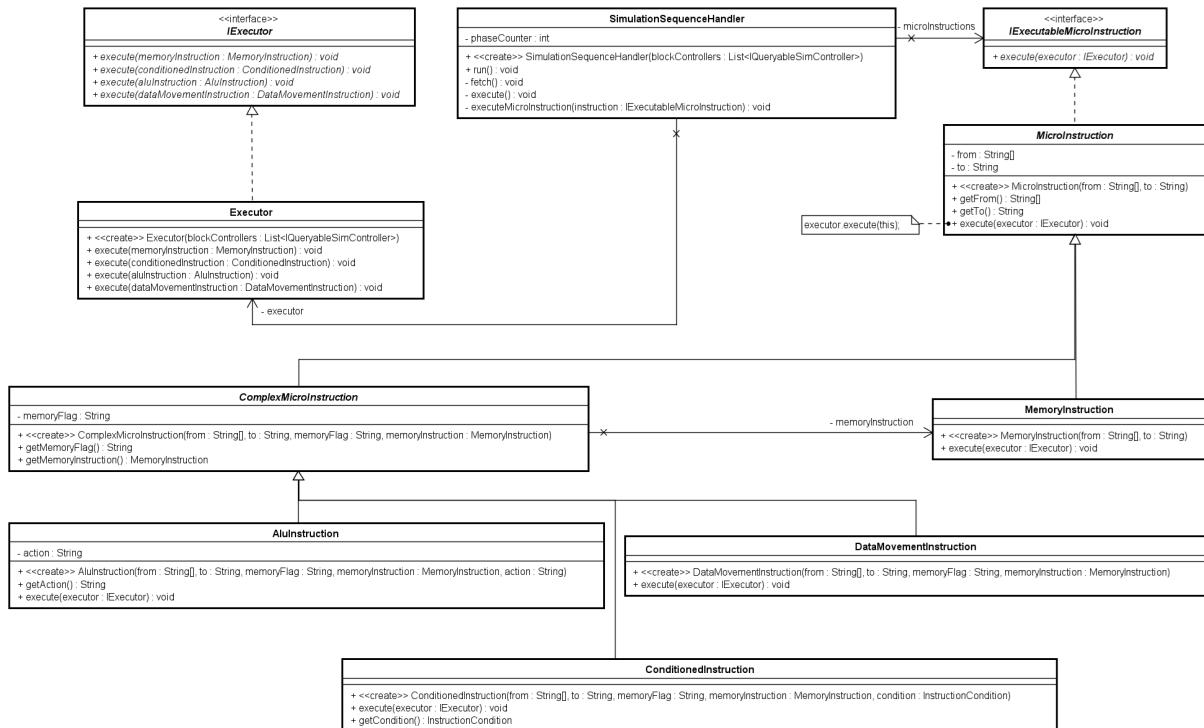


Abbildung 2.16: Klassendiagramm des kombinierten Command- und Visitor-Patterns für die Ausführung von MicroInstructions

Abbildung 2.16 zeigt die Ausführung von *MicroInstructions* *⟨CL20⟩* mithilfe eines Command- und Visitor-Patterns. Jeder Ausführungsschritt wird dabei durch eine *ComplexMicroInstruction* *⟨CL23⟩* des entsprechenden Typs gekapselt. Die Ausführung wird vom *SimulationSequenceHandler* *⟨CL8⟩* initiiert, welcher dann den *Executor* *⟨CL7⟩* gekapselt als *IExecutor* als Visitor an die MicroInstructions übergibt, indem er die *execute()*-Methode der *IExecutableMicroInstruction* aufruft.

PRAXIS DER SOFTWAREENTWICKLUNG

RISCJ Blockits

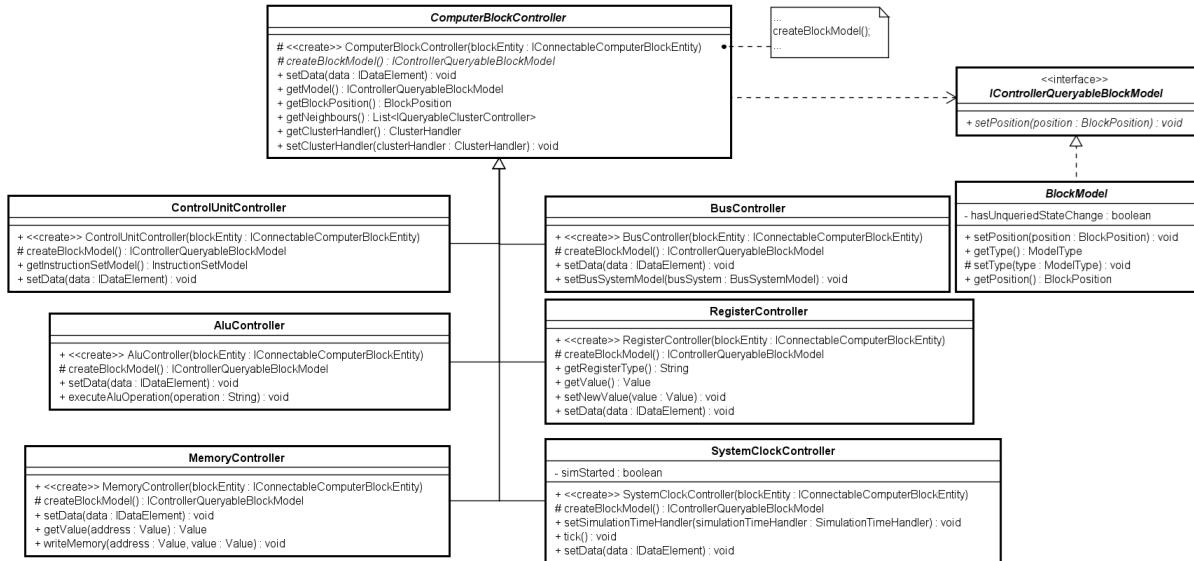


Abbildung 2.17: Klassendiagramm des kombinierten Factory-Method- und Template-Method-Patterns für die Erstellung der Block-Models

Abbildung 2.17 stellt dar, wie die Erzeugung der zu den Controllern gehörigen Models strukturiert ist. Es handelt sich hierbei um eine Kombination einer Factory Method bei `createBlockModel()` und einer Template Method im Konstruktor der Controller, wo die `createBlockModel()`-Methode als von der konkreten Controller-Instanz abhängiger Teilschritt aufgerufen wird. Analog zum hier dargestellten Vorgehen erzeugen auch alle `ComputerBlockEntity`-Instanzen (CL37) ihre zugehörigen Controller.

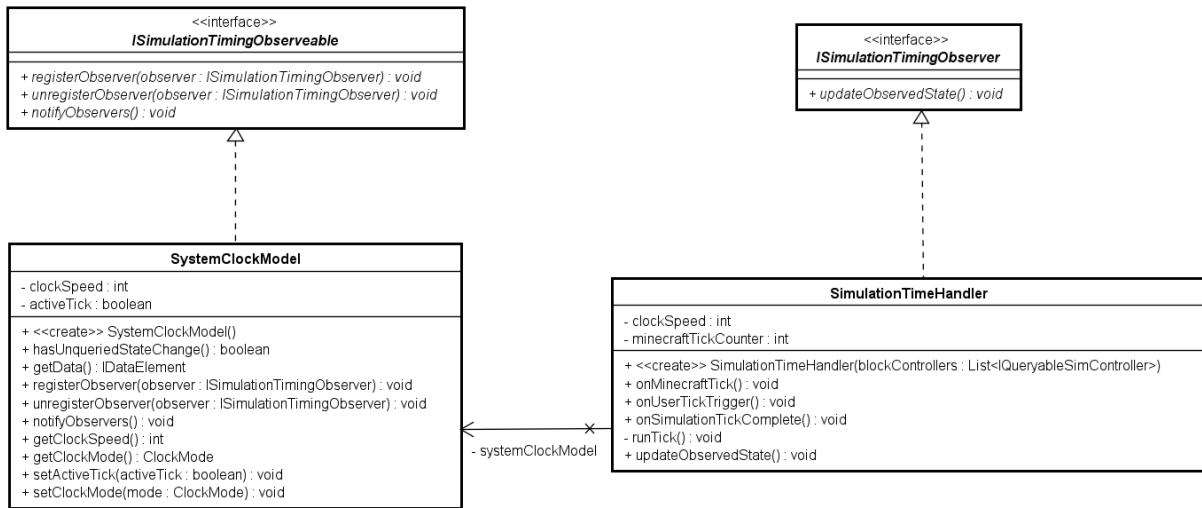


Abbildung 2.18: Klassendiagramm des Observer-Patterns für die Aktualisierung des Zeitmodus der Simulation

Abbildung 2.18 zeigt die Struktur des Observers, der dafür zuständig ist, den Zeitmodus der Simulation im gleichen Zustand zu halten wie den des Models. Der Zeitmodus im Model wird vom *SystemClockController* *⟨CL5⟩* aktualisiert, welcher von der *SystemClockBlockEntity* *⟨CL37⟩* als Folge des User-Inputs aufgerufen wird. Daraufhin benachrichtigt das *SystemClockModel* *⟨CL11⟩* dann sofort seine Observer, insbesondere den *SimulationTimeHandler* *⟨CL9⟩*, damit dieser den nächsten Simulationsticke dann entsprechend des neuen Zeitmodus ausführen kann.

3 Softwareentwurf für interaktive Elemente

3.1 Wichtige Sequenzdiagramme

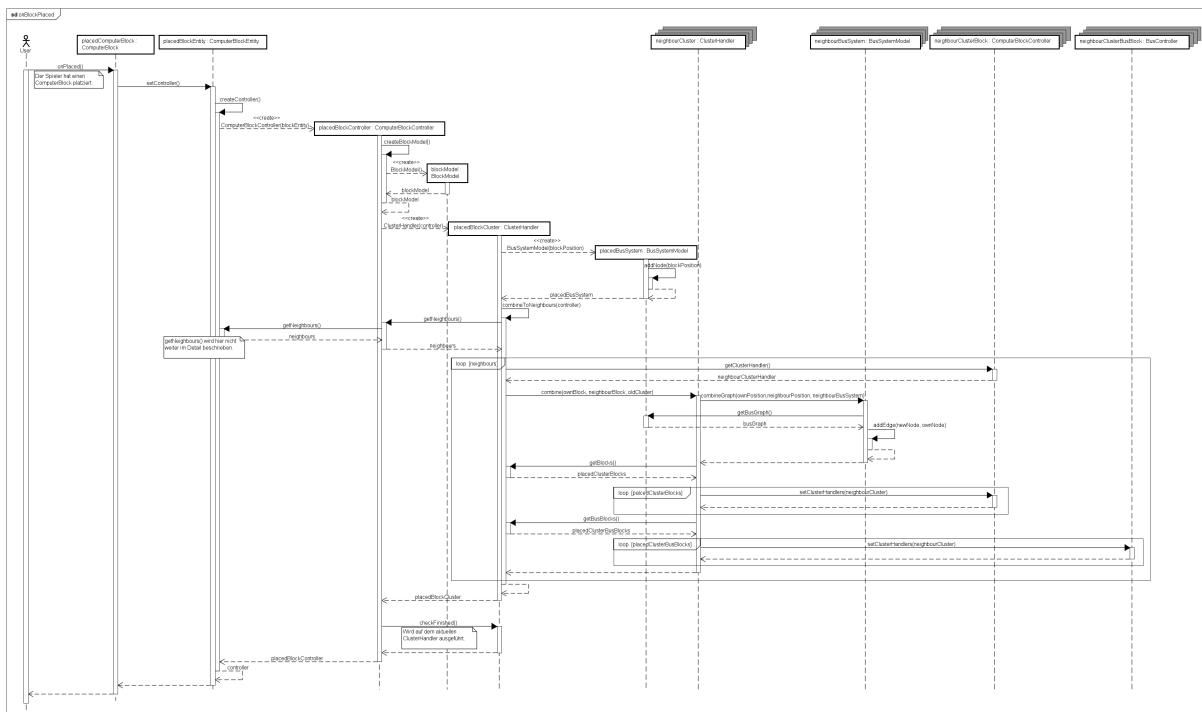


Abbildung 3.1: *ComputerBlock (CL36) platzieren*

Abbildung 3.1 zeigt den Ablauf auf der Server-Seite von Minecraft beim Platzieren eines *ComputerBlocks (CL36)*. Als Erstes wird, wenn der Spieler einen Block platziert, die *onPlaced()* Methode aufgerufen von Minecraft. Beginnend wird ein *ComputerBlockController (CL5)* erstellt. Dieser initialisiert dann das *BlockModel (CL11)* und dann einen *ClusterHandler (CL6)*. Der *ClusterHandler (CL6)* verbündet sich mit benachbarten *ClusterHandlern (CL6)*, die mit Bus-Blöcken mit ihm verbundenen sind. Zum Kombinieren gehört auch die Kombination der Graphen im *BusSystemModel (CL28)*, die den Computer räumlich abbilden. Die Methode *checkFinished* wird genauer in Abbildung 3.2 beschrieben.

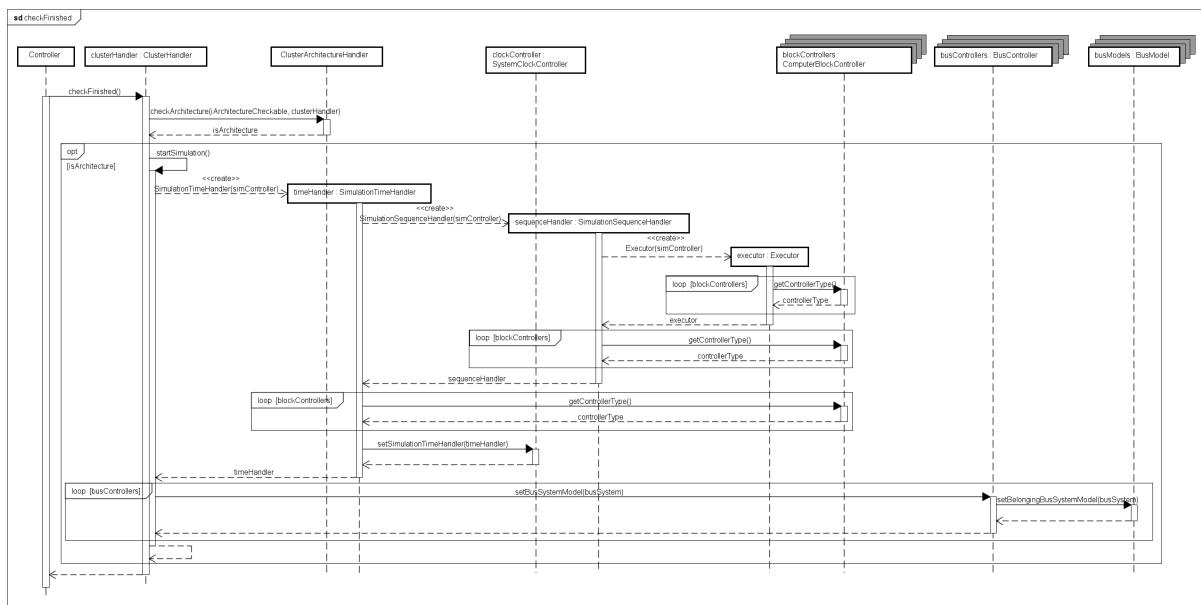


Abbildung 3.2: Cluster auf Vollständigkeit prüfen und Simulation erstellen

Abbildung 3.2 zeigt die Ausführung der Methode `checkFinished`. In dieser wird zunächst mit der Methode `checkArchitecture` überprüft, ob das Cluster eine valide Architektur bildet. Falls das Cluster eine valide Architektur bildet, werden alle für die Simulation benötigten Klassen erstellt und die für die Klassen relevanten `ComputerBlockController` (`CL5`) abgespeichert.

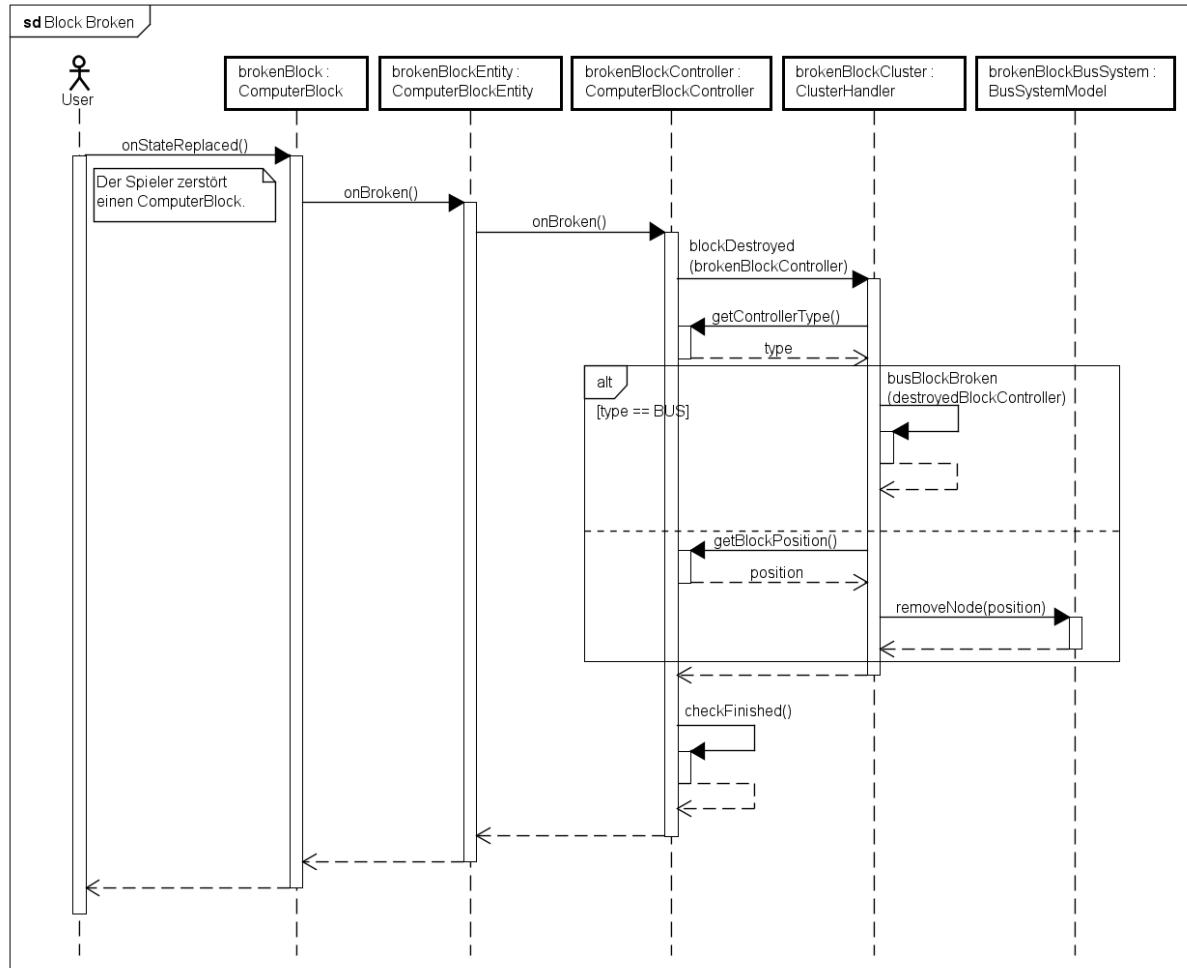
Abbildung 3.3: *ComputerBlock* $\langle CL36 \rangle$ zerstört

Abbildung 3.3 zeigt das Sequenzdiagramm, welches den serverseitigen Ablauf beim Zerstören eines *ComputerBlocks* $\langle CL36 \rangle$ darstellt. Minecraft ruft dafür die `onStateReplaced()` Methode auf. Der Aufruf wird dann weitergegeben an den *ComputerBlockController* $\langle CL5 \rangle$, welcher die Zerstörung dann ausführt. Dies beinhaltet das Entfernen des Blocks aus dem *ClusterHandler* $\langle CL6 \rangle$ und dem Graphen im *BusSystemModel* $\langle CL28 \rangle$. Dabei wird unterschieden zwischen einem *BusBlock* und anderen *ComputerBlocks* $\langle CL36 \rangle$. Falls ein *BusBlock* zerstört wurde, wird die Methode `busBlockBroken()` im *ClusterHandler* $\langle CL6 \rangle$ aufgerufen. Diese Methode wird genauer in Abbildung 3.4 beschrieben.

PRAXIS DER SOFTWAREENTWICKLUNG

RISCJ Blockits

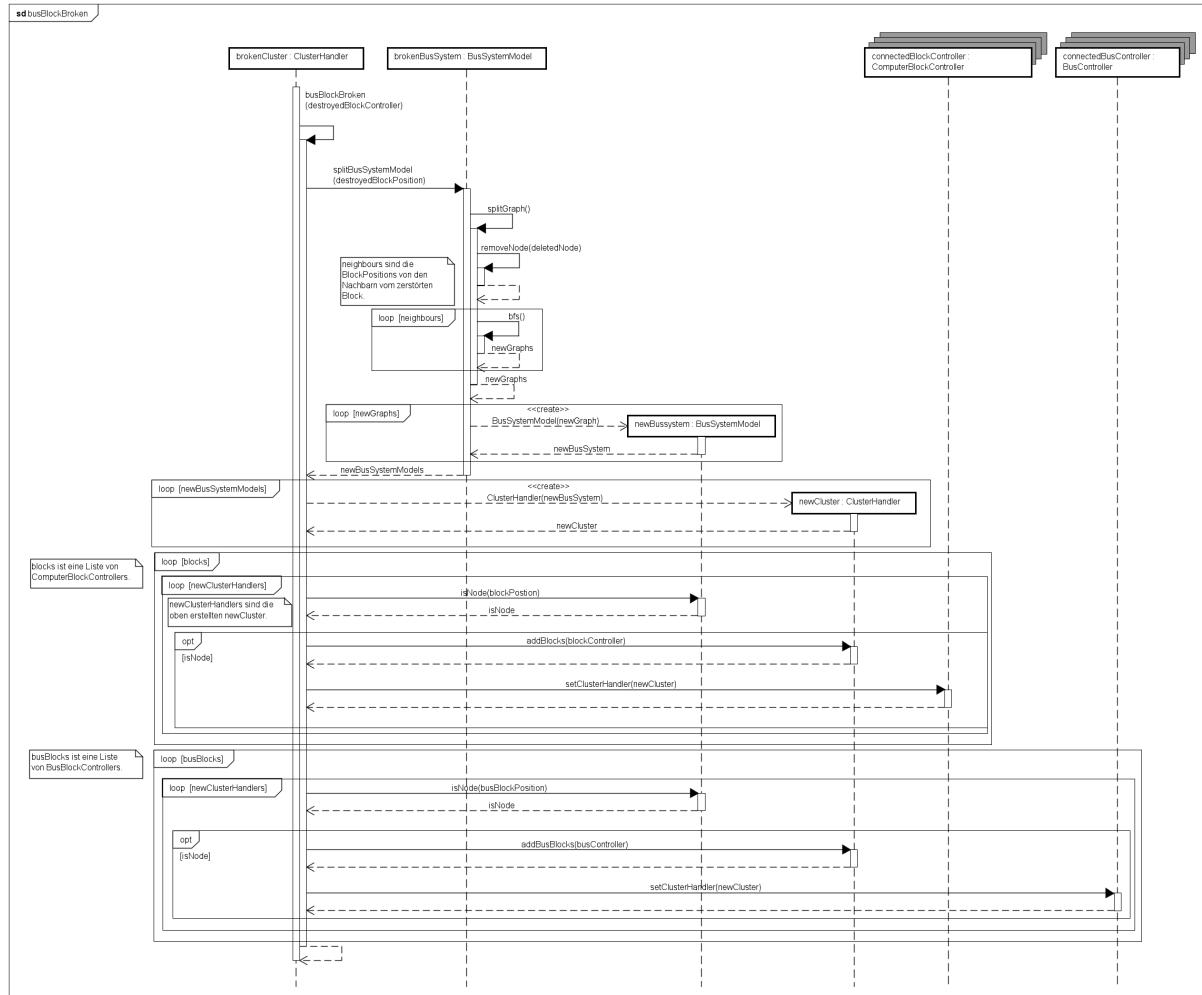


Abbildung 3.4: *BusBlock* zerstört

Abbildung 3.4 zeigt das Sequenzdiagramm, welches den Ablauf der `busBlockBroken()` Methode darstellt. Diese trennt, falls der Bus die letzte Verbindung zwischen zwei Teilgraphen war, ein bestehendes Cluster in mehrere Cluster auf. Dabei muss auch das `BusSystemModel` (CL28) aufgespalten werden. Anschließend müssen die Blöcke und BusBlöcke informiert werden, dass sie jetzt Teil eines anderen Clusters sind.

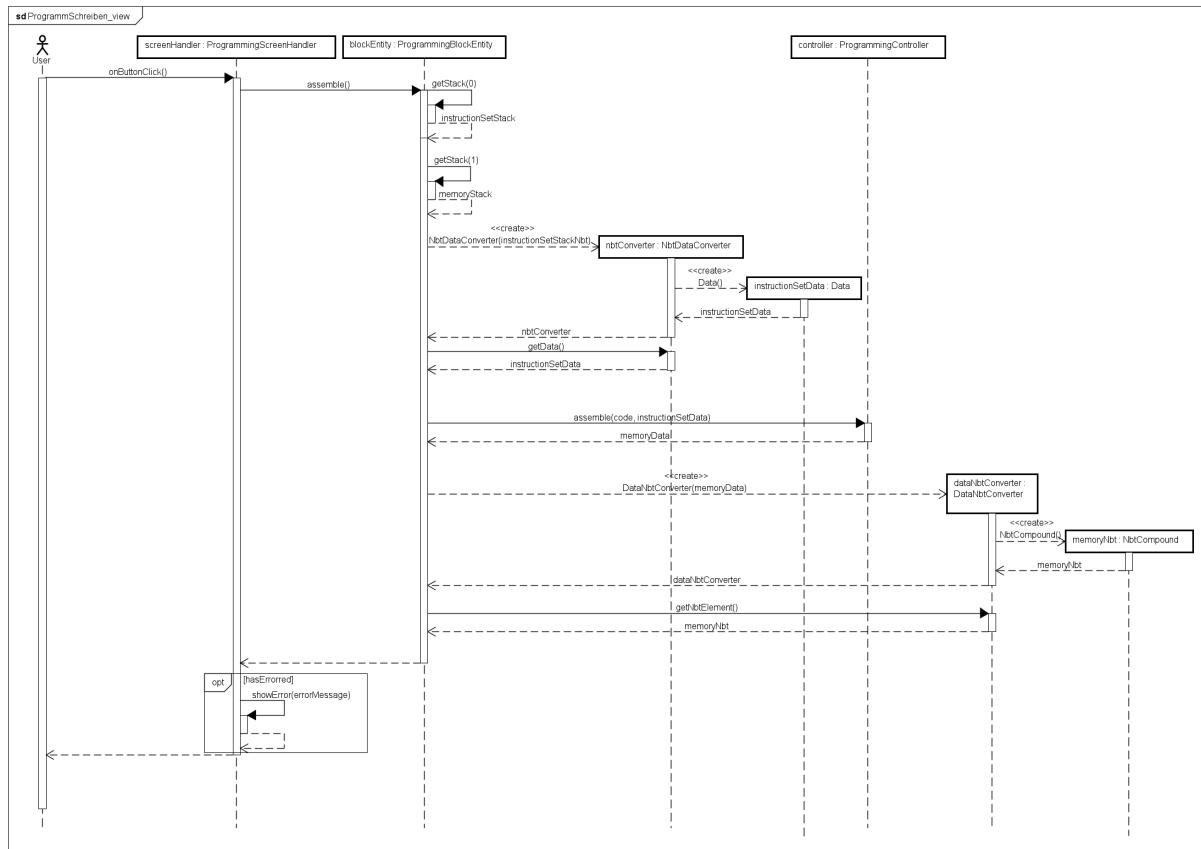


Abbildung 3.5: Programm schreiben (View)

Abbildung 3.5 zeigt das detaillierte Sequenzdiagramm für den Ablauf nach Betätigung des "Assemble" Knopfs im Programm-Editor-GUI. Hierbei werden im *ProgrammingBlockEntity* $\langle CL40 \rangle$ die Nbt-Daten der enthaltenen Items ausgelesen und in ein eigenes Datenformat umgewandelt. Der Ablauf innerhalb der *ProgrammingBlockController::assemble*-Methode wird in Abbildung 3.6 beschrieben. Nach dem Assembeln werden die Daten des beschriebenen Speichers abgespeichert. Falls Fehler aufgetreten sein sollten, werden diese vom *ProgrammingScreenHandler* $\langle CL41 \rangle$ angezeigt.

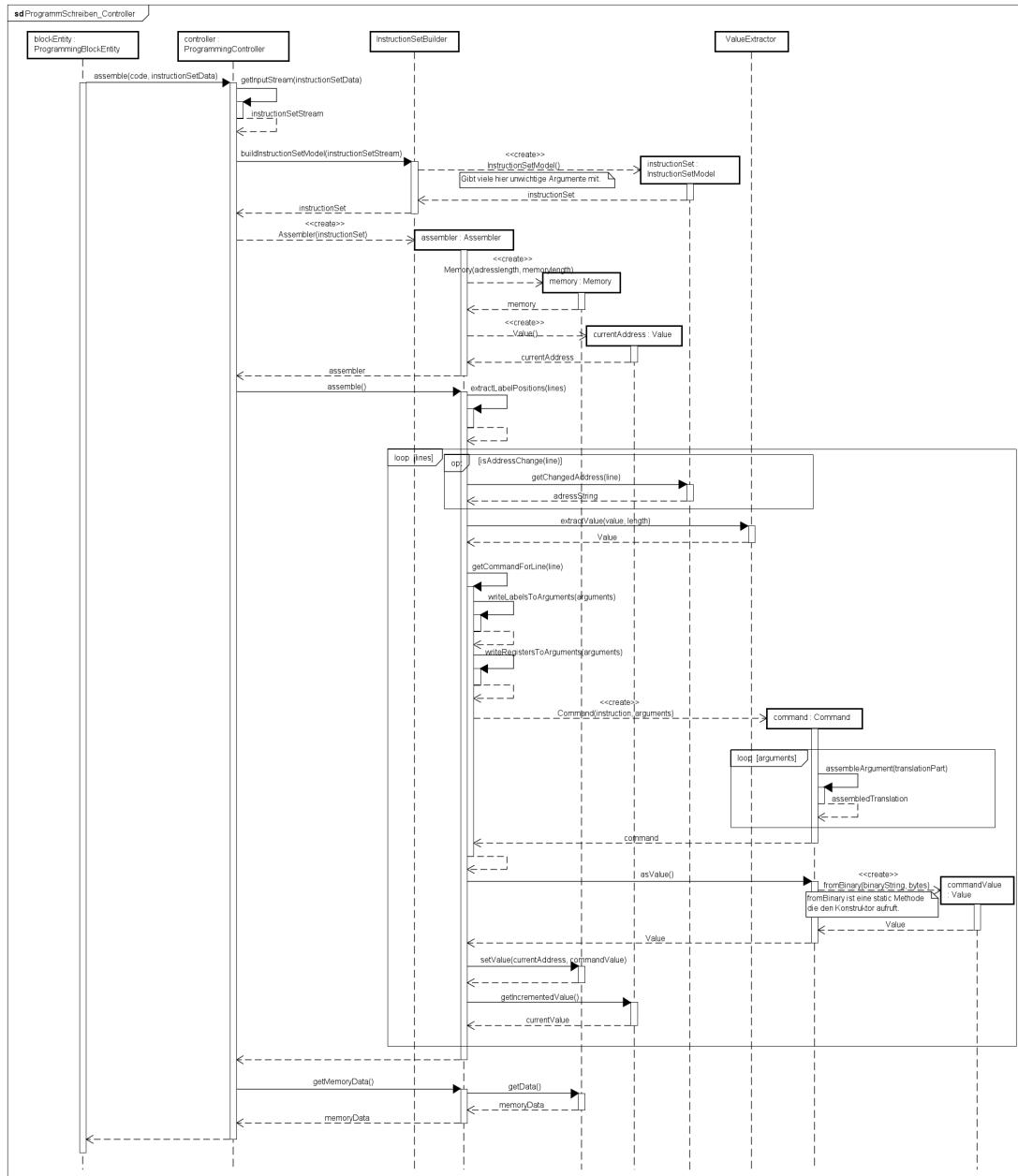


Abbildung 3.6: Programm schreiben (Controller)

Abbildung 3.6 zeigt das detaillierte Sequenzdiagramm für den Ablauf nach Aufruf der "assemble" Methode des *ProgrammingBlockController*'s. Anschließend wird der *DataStreamEntry* (CL12), der das instructionSet-JSON enthält, entpackt. Dieser Schritt ist der Länge des Diagramms wegen nicht dargestellt. Nach der Erzeugung des *InstructionSetModel* (CL17) wird der Code im *Assembler* (CL1) assembled. Hier werden zuerst die Labels erkannt und die Adressen, zu denen diese zeigen, berechnet. Anschließend werden in jeden Befehl Argumente und Registerübersetzungen eingefügt. Dann werden die Befehle zusammengesetzt und in Values verpackt. Diese *Values* (CL27) werden schließlich in den *Memory* (CL26) geschrieben.

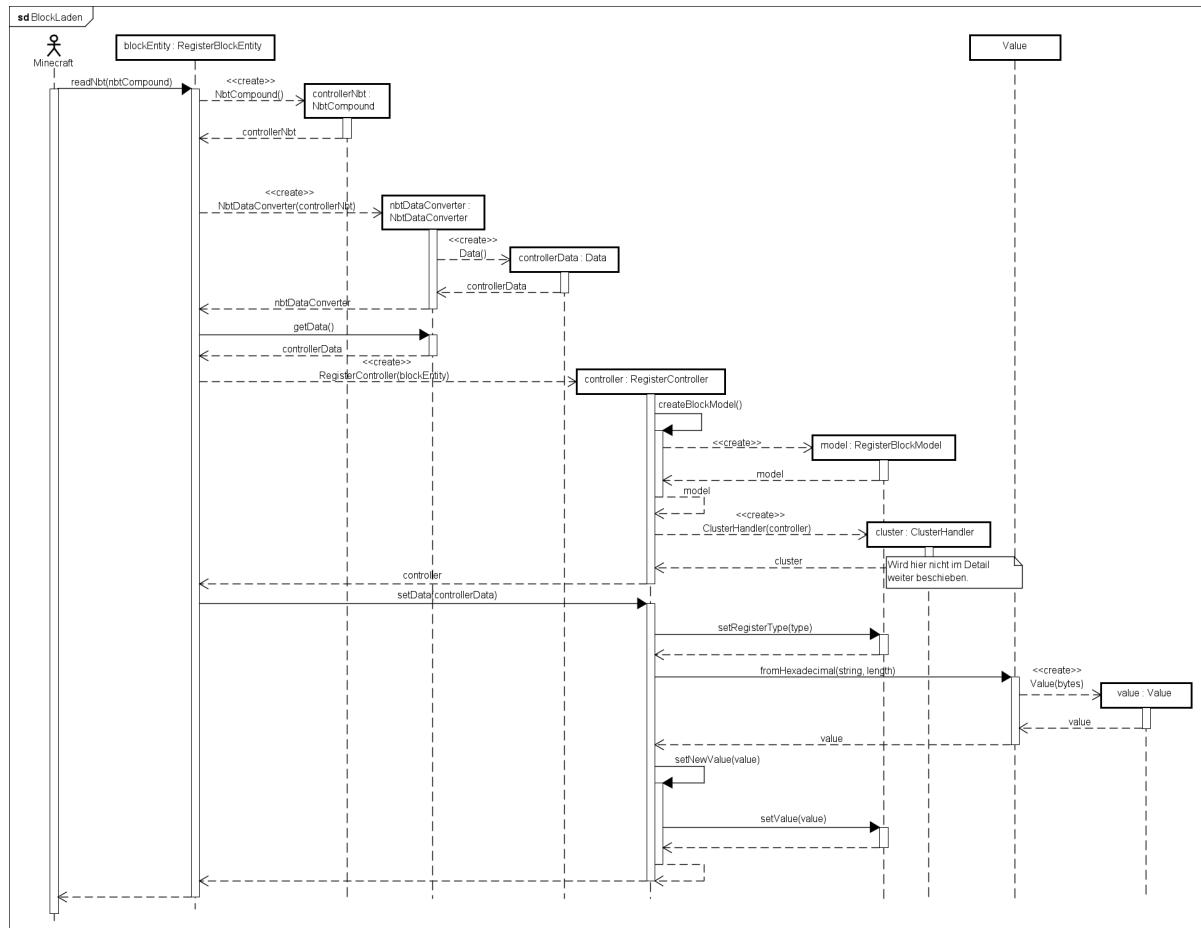


Abbildung 3.7: Block Laden

Abbildung 3.7 zeigt das detaillierte Sequenzdiagramm des Ablaufs beim Laden eines Blocks, hier am konkreten Beispiel der *RegisterBlocks*. Dies geschieht, wenn eine Minecraft-Welt betreten wird oder ein Spieler sich nah genug an einem vorher gespeicherten Block befindet. Hierzu wird einer *BlockEntity* der von ihm gespeicherten *NbtCompound* übergeben, aus welchem es Daten lädt. Die *RegisterBlockEntity* konvertiert einen Teil der Nbt-Daten in eine *Data*-Instanz, mit welcher sie dann einen erstellten *RegisterBlockController* initialisiert. Der Controller konfiguriert mit den Daten das Model.

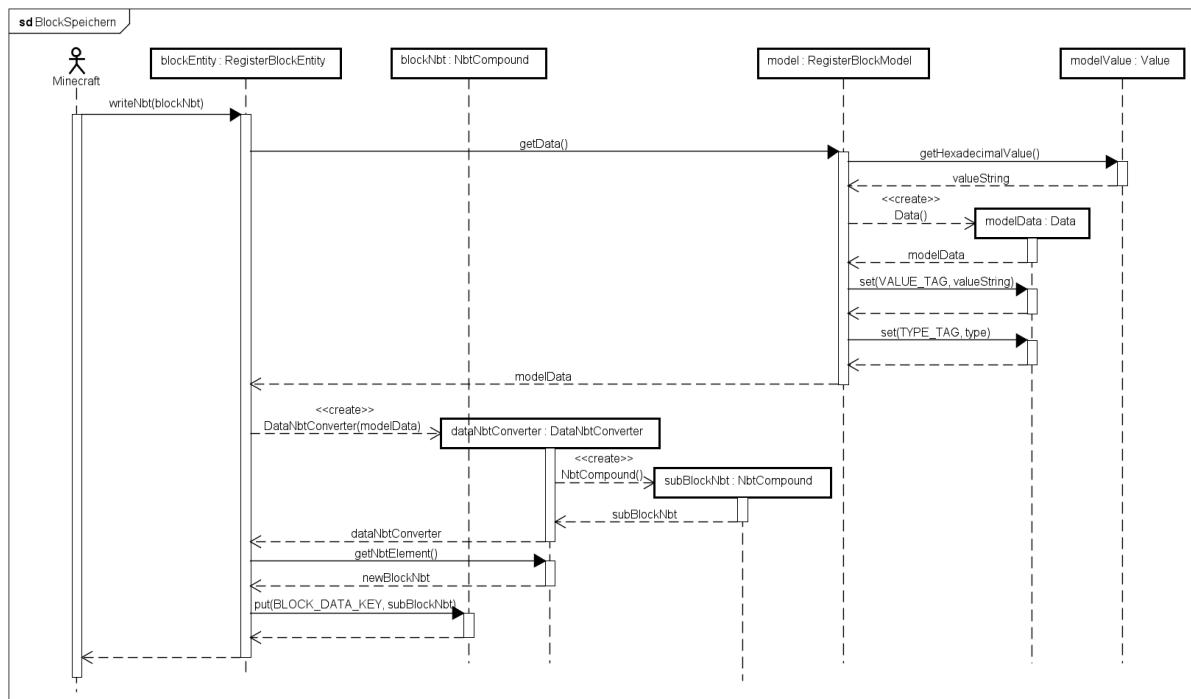


Abbildung 3.8: Block Speichern

Abbildung 3.8 zeigt das detaillierte Sequenzdiagramm des Ablaufs beim Speichern eines Blocks, hier am konkreten Beispiel der *RegisterBlocks*. Dies geschieht, wenn die Minecraft-Welt verlassen wird oder kein Spieler mehr in der Nähe des Blocks ist. Hierzu bekommt eine *BlockEntity* ein *NbtCompound*, in welchen sie ihre relevanten Daten speichert. Die zu speichernden Daten ruft die *BlockEntity* beim Model ab, konvertiert sie zu Nbt, und schreibt sie in den erhaltenen Compound.

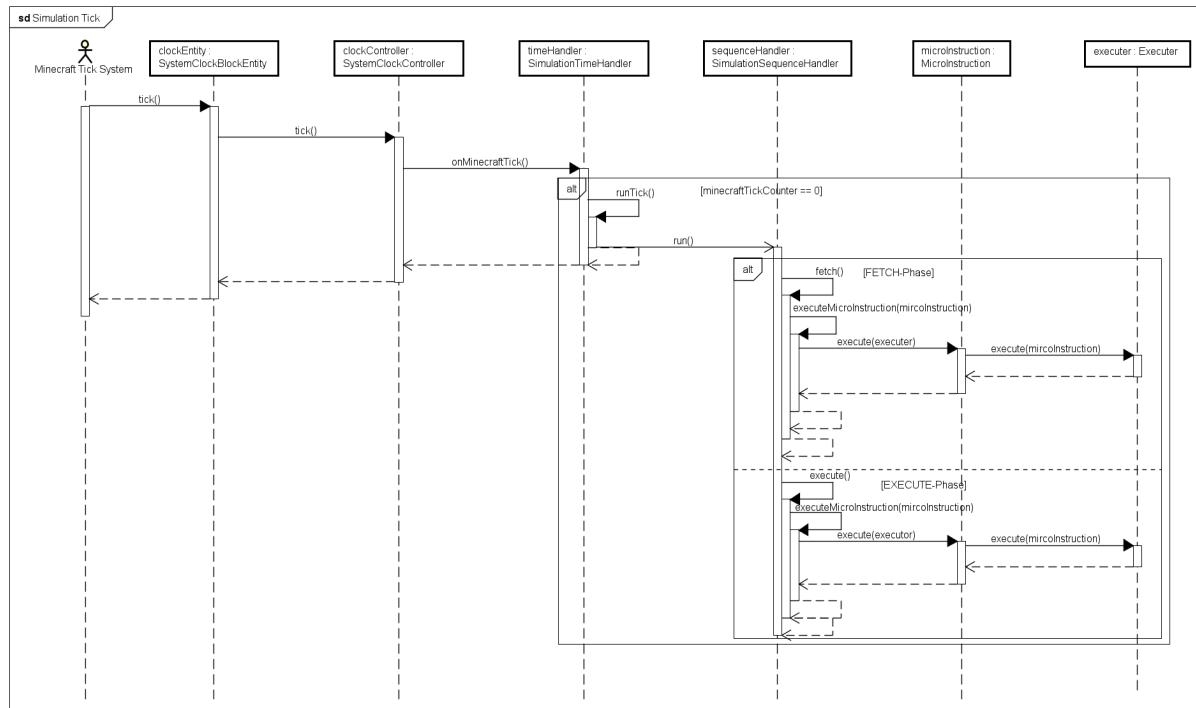


Abbildung 3.9: Ausführung eines Simulations-Ticks

Abbildung 3.9 zeigt den Ablauf nach dem Auslösen eines Ticks durch Minecraft. Zunächst löst Minecraft die `tick()`-Methode der `SystemClockBlockEntity` `<CL37>` aus. Diese reicht den Aufruf an ihren Controller weiter, welcher den `SimulationTimeHandler` `<CL9>` aufruft. Dieser entscheidet abhängig von der im Tick-Modus vom Nutzer gesetzten Frequenz, ob ein Minecraft-Tick ausgeführt wird, oder ob es sich um einen Warte-Tick handelt und nur der Zähler der Warte-Ticks inkrementiert wird. Im Falle eines Warte-Ticks wird die Kontrolle wieder zurück an Minecraft übergeben. Andernfalls wird der `SimulationSequenceHandler` `<CL8>` über die `run()`-Methode aufgerufen und entscheidet dann abhängig von der aktiven Ausführungsphase Fetch oder Execute, welche `IExecutableMicroInstruction` ausgeführt wird. Diese wird dann nach dem Visitor-Pattern, welches in Abbildung 2.16 gezeigt ist, ausgeführt.

4 Eigene Datenstrukturen

4.1 Befehlssatz im .json-Format

Der Befehlssatz, mit dem der Computer arbeitet, ist in einer .json-Datei spezifiziert, die alle Übersetzungen von Assembler in Binär-Form und von Binär-Form in von uns ausführbare Instruktionen definiert.

4.1.1 Mikroinstruktionsformat

Die Übertragung von Mikroinstruktionen aus der .json-Datei erfordert ein spezifisches Format, damit der *MicroInstructionDeserializer* $\langle CL21 \rangle$ eine korrekte Übersetzung in die Code-Repräsentation der Mikroinstruktionsarten sicherstellen kann.

Es ist festzuhalten, dass in alle aufrufbaren Mikroinstruktionsformate Register durch ihre vorgegebenen Bezeichner aus dem Befehlssatz oder Konstanten aus den Parametern durch Angabe in eckigen Klammern eingesetzt werden können.

AluInstruction

Eine *AluInstruction* führt einen Teilschritt einer *MicroInstruktion* aus, bei dem eine ALU-Aktion ausgeführt werden muss. Sie ist immer nach dem Schema [“<operation>“, “<alu-dest>“, “<alu-origin 1>“, “<alu-origin 2>“, “<memory flag>“, “<MemoryInstruction>“] aufgebaut.

DataMovementInstruction

Eine *DataMovementInstruction* ist für das Lesen und Schreiben von Daten von einem Register zu einem anderen zuständig und entspricht somit für die Visualisierung auch einer Bus-Aktivierung. Die Struktur für diese *MicroInstruction* ist [“<destination>“, “<origin>“, “<memory flag>“, “<MemoryInstruction>“].

ConditionedInstruction

Um Branch- und Jump-Befehle zu ermöglichen, existiert zudem eine bedingte Mikroinstruktion, welche bei Erfüllung einer angegebenen Bedingung eine Datenbewegung ausführt und sonst zur nächsten Operation übergeht. Das Format für bedingte Operationen lautet [“IF“, [“<comparator 1>“, “<comparator 2>“, “<comparing operation>“], [“<then destination>“, “<then origin>“], “<memory flag>“, “<MemoryInstruction>“].

MemoryInstruction

Eine *MemoryInstruction* ist für Bewegungen zwischen Speicher und eventuellen Speicher-adjazenten Registern, bei MIMA beispielsweise dem SDR, vorgesehen. Sie kann daher nicht einzeln aufgerufen werden, sondern ist ein Bestandteil der anderen Mikroinstruktionsformate. Der Aufbau einer Speicher-Mikroinstruktion muss immer dem Format [“<destination>“, “<origin>“] entsprechen.

4.1.2 Dateistruktur

Zunächst wird in der Datei der Name des Befehlssatzes spezifiziert und die Länge eines Befehls festgelegt, wie in Abbildung 4.1 gezeigt. Diese Informationen werden im Code nicht benutzt, sondern dienen der Vollständigkeit für Entwickler.

```
1   "name": "MIMA",
2   "instruction_length": 24,
```

Abbildung 4.1: Spezifikation der Basisinformationen des Befehlssatzes

Im folgenden Teil der .json-Datei werden die Register festgelegt, die in diesem Befehlssatz vorliegen. Das in Abbildung 4.2 dargestellte Beispiel für MIMA zeigt, dass dabei eine Unterteilung in Befehlszähler-Register, Ein- und Ausgabe-Register der ALU und sonstige Float- sowie Integer-Register vorgenommen wird. Diese dient der Optimierung von Zugriffen zum Finden des gesuchten Registers für einen bestimmten Zweck.

Anschließend werden, wie in Abbildung 4.3 gezeigt, die Speicherspezifikationen gesetzt. Die Informationen zu Opcode-Längen und Position dienen zur späteren Zuordnung von Instruktionen in Binärform aus dem Speicher auf ihre Repräsentation als Instanz der *Instruction*-Klasse (CL14).

In einem weiteren Abschnitt werden die ALU-Aktionen, die der Befehlssatz anbietet, aufgelistet. Dabei ist zu beachten, dass nur die im für das Projekt manuell reduzierten RISC-V-Befehlssatz

```
1   "registers": {
2     "program_counter": "IAR",
3     "alu": [
4       "X",
5       "Y",
6       "Z"
7     ],
8     "float": {},
9     "integer": {
10       "IR": 1,
11       "EINS": 2,
12       "AKKU": 3,
13       "SAR": 4,
14       "SDR": 5
15     }
16   }
```

Abbildung 4.2: Spezifikation der Register

```
1   "memory": {
2     "word_length": 24,
3     "address_length": 20,
4     "access_delay": 3,
5     "byte_order": "le",
6     "possible_opcode_lengths": [4, 8],
7     "opcode_position": "MOST"
8   }
```

Abbildung 4.3: Spezifikationen für Speicher-Größen

sowie im Beispiel in Abbildung 4.4 für den MIMA-Befehlssatz genutzten Aktionen implementiert sind. Eine Erweiterung der ALU-Befehle setzt daher eine Manipulation des Source-Codes in der *ALUController*-Klasse $\langle CL5 \rangle$ voraus.

```
1   "alu_operations": [
2     "None",
3     "ADD",
4     "RR",
5     "AND",
6     "OR",
7     "XOR",
8     "NEG",
9     "JMN"
10    ]
```

Abbildung 4.4: ALU-Operationen, die im Befehlssatz erlaubt sind

Der folgende Abschnitt der .json-Datei spezifiziert nun die Hol-Phase für Instruktionen in diesem Befehlssatz. Dabei sind die einzelnen Schritte im manuellen Mikroinstruktionsformat definiert. Im Beispiel in Abbildung 4.5 ist die Hol-Phase für MIMA dargestellt.

```
1   "fetch": [
2     ["SAR", "IAR", "r", ["<mem_addr>", "SAR"]],
3     ["X", "IAR", "r"],
4     ["Y", "EINS", "r"],
5     ["ADD", "Z", "X", "Y"],
6     ["IAR", "Z", "r", ["SDR", "<mem_data>"]],
7     ["IR", "SDR", ""]
8   ]
```

Abbildung 4.5: Ablauf der Holphase

Schließlich werden im letzten Abschnitt der Datei die Instruktionen, die der Befehlssatz anbietet, spezifiziert. Hierfür wird eine Zuordnung des Befehlsworts zu den zugehörigen Informationen bereitgestellt. Jeder Befehl muss darin seine Argumente, seinen Opcode, seine Ausführung als Folge von Mikroinstruktionen im korrekten Format und eine Übersetzungsvorschrift in Binärform definieren. Abbildung 4.6 zeigt einen Auszug aus der MIMA-Spezifikation, der den Load-Constant-Befehl und den Add-Befehl beinhaltet.

```
1   "instructions": {
2     "LDC": {
3       "arguments": "[[const]]",
4       "opcode": "0000",
5       "execution": [
6         ["AKKU", "[const]", ""]
7       ],
8       "translation": [
9         "0000",
10        "[const]<20>"
11      ]
12    },
13    "ADD": {
14      "arguments": "[addr]",
15      "opcode": "0011",
16      "execution": [
17        ["SAR", "[addr]", "r", ["<mem_addr>", "SAR"]],
18        ["X", "AKKU", "r"],
19        [ "", "", "r", ["SDR", "<mem_data>"]],
20        [ "Y", "SDR", ""],
21        [ "ADD", "Z", "X", "Y", ""],
22        [ "AKKU", "Z", ""]
23      ],
24      "translation": [
25        "0011",
26        "[addr]<20>"
27      ]
28    },
29  }
```

Abbildung 4.6: Spezifikation von Befehlen

5 Glossar

Assembler: Ein Assembler, ist ein Computerprogramm, das Assemblersprachcode in Maschinencode übersetzt. Assemblersprache ist eine Low-Level-Programmiersprache, die eng mit dem Maschinencode verwandt ist, aber für Menschen leichter zu lesen und zu schreiben ist.¹

assembeln: Als "assembeln" (engl. to assemble) bezeichnet man den Vorgang im Assembler, bei dem die Attribute und Befehle in für den Computer verständlichen Maschinencode übersetzt werden.

Befehlssatz: Der Befehlssatz eines Prozessors ist in der Rechnerarchitektur die Menge der Maschinenbefehle, die ein bestimmter Prozessor ausführen kann. Je nach Prozessor variiert der Umfang des Befehlssatzes zwischen beispielsweise 33 und über 500 Befehlen. CISC-Prozessoren haben tendenziell größere Befehlssätze als RISC-Prozessoren.²

Block: Ein Block ist ein besonderer Gegenstand, den man in der Minecraft-Welt platzieren kann. Nahezu die gesamte Spielwelt besteht aus Blöcken.³

Cluster: Ein Cluster ist in der Mod mehrere aneinander platzierte Bus Blöcke mit ihren benachbarten Computerblöcken.

Fabric: Ein Modloader (Programm), dass es erlaubt Modifikationen für Minecraft in das Spiel zu laden. Es bietet eine separate API, welche die Kommunikation mit dem Spiel ermöglicht. Fabric wurde zuerst als Hobby-Projekt entwickelt, da bisherige Modloader eine zu zeitaufwendige Entwicklung benötigten.⁴

Inventar: Im Überlebensmodus ist das Inventar der Speicher des Spielers für Blöcke und sonstige Gegenstände, sozusagen ein Rucksack, den man immer bei sich trägt. Das Inventar hat ein begrenztes Fassungsvermögen. Im Kreativmodus ist das Inventar kein Speicher, sondern eine Auswahl fast aller Blöcke und sonstiger Gegenstände in unbegrenzter Menge.⁵

Item: Ein Gegenstand (engl. Item) ist alles, was man in sein Inventar aufnehmen und in der Hand halten kann. Ein Block ist ein besonderer Gegenstand, den man in der Welt platzieren kann (z.B. Erde oder eine Werkbank). Daneben gibt es noch einige Gegenstände, die beim Platzieren zu einem beweglichen Objekt werden, z.B. ein Boot.⁶

¹Quelle: <https://techwatch.de/blog/understanding-the-basics-what-is-an-assembler-and-how-does-it-work/>

²Quelle: <https://de.wikipedia.org/wiki/Befehlssatz>

³Vgl.: <https://minecraft.fandom.com/de/wiki/Block>

⁴Quelle: <https://ftb.fandom.com/wiki/Fabric>

⁵Quelle: <https://minecraft.fandom.com/de/wiki/Inventar>

⁶Quelle: <https://minecraft.fandom.com/de/wiki/Gegenstand>

JSON: Kurz für "JavaScript Object Notation" ist ein kompaktes Datenformat in einer einfach lesbaren Textform für den Datenaustausch zwischen Anwendungen. JSON ist von Programmiersprachen unabhängig. Die Daten können beliebig verschachtelt werden, beispielsweise ist eine indizierte Liste (englisch "array") von Objekten möglich, welche wiederum arrays oder Objekte enthalten.⁷

Kreativmodus: Einer der 5 Modi in Minecraft. Wie der Name schon andeutet, ist der Kreativmodus (engl. Creativemode) speziell für das kunstvolle Erschaffen von umfangreichen, komplexen Bauwerken konzipiert. Dafür bietet er dem Spieler unendlich viele Ressourcen und Unsterblichkeit.⁸

MIMA: Die mikroprogrammierte Minimalmaschine (MIMA) ist ein Lehrmodell zur vereinfachten Darstellung von Mikroprozessoren, basierend auf der Von-Neumann-Architektur, welche von Tamim Asfour am Karlsruher Institut für Technologie entwickelt wurde.⁹

Minecraft: Minecraft ist ein Open-World-Computerspiel, das ursprünglich vom Schweden Markus "Notch" Persson erschaffen wurde. Siehe <https://minecraft.fandom.com/de/wiki/Minecraft>¹⁰

Minecraft-Objekte/Entity: Objekte (engl. Entities) sind Gegenständen, spezielle Blöcke und Lebewesen in Minecraft. Sie können beweglich sein, individuell Daten speichern, Items aufnehmen oder eine Benutzeroberfläche bereitstellen. Damit unterscheiden sie sich von den normalen statischen Blöcken, die keine individuellen Eigenschaften haben.

Minecraft-Tick: Minecraft funktioniert mit Ticks. Standardmäßig gibt es 20 Ticks pro Sekunde. Jeden Tick werden die Blöcke in der Minecraft Welt aktualisiert. Ein Minecraft Tag dauert 24000 Ticks.¹¹

Mod: Als Modifikation (Abkürzung Mod) wird alles bezeichnet, das den Spielinhalt von Minecraft verändert.¹²

Nbt: kurz für "Named Binary Tag" ist das interne Datenformat Minecrafts. Hier werden, ähnlich zum JSON Format, tiefer-liegende Nbt-Daten anhand eines Schlüssels abgespeichert. Somit können diese strukturiert wieder abgerufen werden.¹³

Rezepte: Rezepte geben an, mit welchen Gegenständen ein Block oder Gegenstand in Minecraft hergestellt werden kann.

⁷Quelle: https://de.wikipedia.org/wiki/JavaScript_Object_Notation

⁸Vgl.: <https://minecraft.fandom.com/de/wiki/Kreativmodus>

⁹Vgl.: https://de.wikipedia.org/wiki/Mikroprogrammierte_Minimalmaschine

¹⁰Quelle: <https://minecraft.fandom.com/de/wiki/Minecraft>

¹¹Quelle: <https://minecraft.fandom.com/de/wiki/Tick>

¹²Quelle: <https://minecraft.fandom.com/de/wiki/Modifikation>

¹³Quelle: <https://minecraft.fandom.com/de/wiki/NBT>

RISC-V: RISC-V ist eine Befehlssatzarchitektur, die sich auf das Designprinzip des Reduced Instruction Set Computers (RISC) stützt. Das Designziel von RISC ist der Verzicht auf einen komplexen Befehlssatz hin zu einfach zu dekodierenden und schnell auszuführenden Befehlen.

¹⁴

Schnellzugriffsleiste (engl. Hotbar) : Neun Slots des Inventars bilden die Schnellzugriffsleiste, die stets am unteren Fensterrand angezeigt wird. Einer der Slots der Schnellzugriffsleiste ist der aktuelle Slot. Den Inhalt dieses Slots hält man in der Hand.

Server-Side/Client-Side (logisch/physikalisch): Minecraft verwendet das Client-Server-Modell, d.h. die Benutzer installieren den Spiel-Client und verbinden sich mit einem Server, um das Spiel zu spielen. Fabric ermöglicht Mods, die entweder auf den Minecraft-Client oder den Minecraft-Server abzielen, aber auch auf beide gleichzeitig. Das Konzept von Client/Server in Minecraft ist mehrdeutig und kann sich entweder auf die physische oder die logische Seite beziehen. Die Begriffe Client/Server können verwendet werden, um zwischen den verschiedenen Distributionen von Minecraft (dem Minecraft-Client und einem dedizierten Minecraft-Server) zu unterscheiden, die als physische Seiten bezeichnet werden. Ein Minecraft-Client hostet jedoch seinen eigenen integrierten Server für Einzelspieler- und LAN-Sitzungen, was bedeutet, dass ein Minecraft-Client auch Serverlogik enthält. Daher kann der Begriff Client/Server auch verwendet werden, um Teile der Spiellogik zu unterscheiden, die als logische Seiten bezeichnet werden. Für beide Arten von Seiten gibt es einen Client und einen Server. Ein logischer Client ist jedoch nicht gleichbedeutend mit einem physischen Client, und ein logischer Server ist auch nicht gleichbedeutend mit einem physischen Server. Ein logischer Client wird stattdessen von einem physischen Client gehostet, und ein logischer Server wird entweder von einem physischen Server oder einem physischen Client gehostet.

Slot: Ein Slot in Minecraft ist ein Platz in der GUI, an den Items gelegt werden können. Das Inventar und Kisten bestehen aus Slots.

Überlebensmodus: Einer der 5 Modi in Minecraft. Der Schwerpunkt des Spiels liegt beim Überlebensmodus (engl. Survivalmode), wie der Name schon sagt, auf dem Überleben des Spielers.

¹⁷

¹⁴vgl.: <https://de.wikipedia.org/wiki/RISC-V> und https://de.wikipedia.org/wiki/Reduced_Instruction_Set_Computer

¹⁵Quelle: [#Inventar_im_\%C3\%9Cberlebensmodus](https://minecraft.fandom.com/de/wiki/Inventar)

¹⁶Quelle: <https://fabricmc.net/wiki/tutorial:side> übersetzt mit DeepL

¹⁷Vgl.: <https://minecraft.fandom.com/de/wiki/Überlebensmodus>

6 Anhang

6.1 Komplettes Klassendiagramm

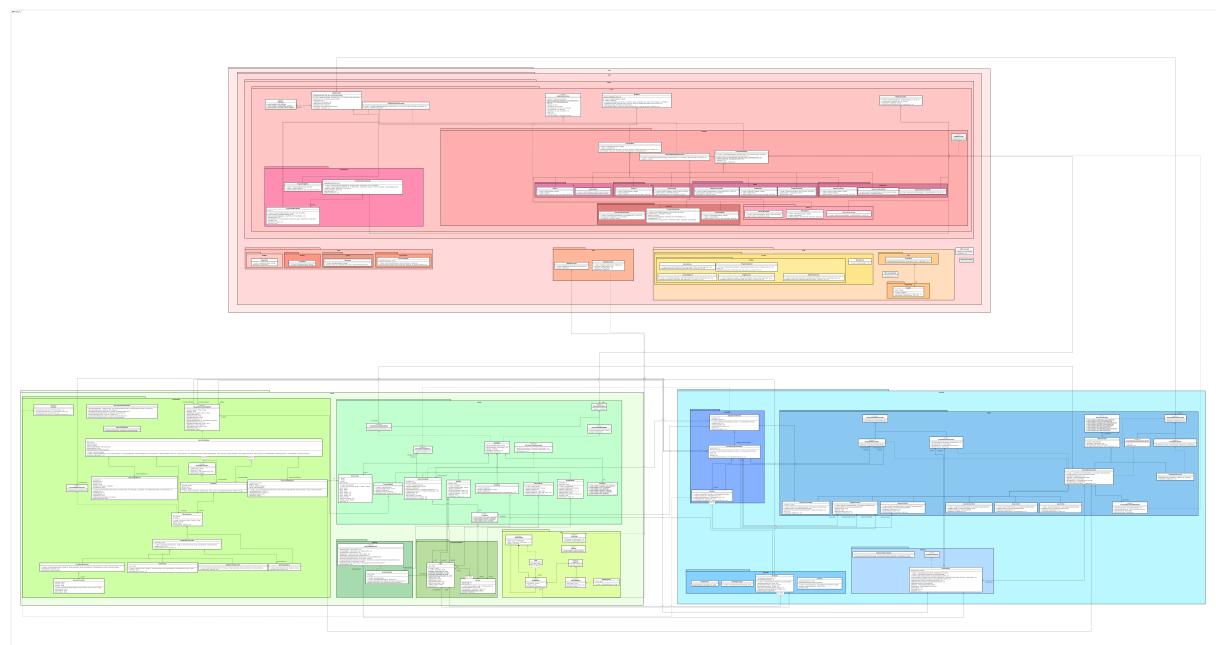


Abbildung 6.1: Klassendiagramm gesamt

Leider ist das Diagramm nur in einem guten PDF-Viewer lesbar, da das Bild sehr hochauflösend und zu groß für die meisten PDF-Viewer ist.

6.2 Komplettes MIMA.json

```
{  
  "name": "MIMA",  
  "instruction_length": 24,  
  
  "registers": {  
    "program_counter": "IAR",  
    "alu": [  
      "X",  
      "Y",  
      "Z"  
    ]  
  }  
}
```

```
],  
    "float": {},  
    "integer": {  
        "IR": 1,  
        "EINS": 2,  
        "AKKU": 3,  
        "SAR": 4,  
        "SDR": 5  
    }  
},  
    "memory": {  
        "word_length": 24,  
        "address_length": 20,  
        "access_delay": 3,  
        "byte_order": "le",  
        "possible_opcode_lengths": [4, 8],  
        "opcode_position": "MOST"  
    },  
    "alu_operations": [  
        "None",  
        "ADD",  
        "RR",  
        "AND",  
        "OR",  
        "XOR",  
        "NEG",  
        "JMN"  
    ],  
    "fetch": [  
        ["SAR", "IAR", "r", ["<mem_addr>", "SAR"]],  
        ["X", "IAR", "r"],  
        ["Y", "EINS", "r"],  
        ["ADD", "Z", "X", "Y"],  
        ["IAR", "Z", "r", ["SDR", "<mem_data>"]],  
        ["IR", "SDR", ""]  
    ],  
    "address_change": {  
        " *\\* *= *(?<address>\\w+) *": "[address]"  
    },
```

```
"program_start_label":  
    "START",  
"data_storage_keywords": {  
    " *DS * (?<value>(?:(:0x)|(:0b))\\d+)" : "[value]"  
},  
"instructions": {  
    "LDC": {  
        "arguments": ["[const]"],  
        "opcode": "0000",  
        "execution": [  
            ["AKKU", "[const]", ""]  
        ],  
        "translation": [  
            "0000",  
            "[const]<20>"  
        ]  
    },  
    "ADD": {  
        "arguments": ["[addr]"],  
        "opcode": "0011",  
        "execution": [  
            ["SAR", "[addr]", "r", ["<mem_addr>", "SAR"]],  
            ["X", "AKKU", "r"],  
            ["", "", "r", ["SDR", "<mem_data>"]],  
            ["Y", "SDR", ""],  
            ["ADD", "Z", "X", "Y", ""],  
            ["AKKU", "Z", ""]  
        ],  
        "translation": [  
            "0011",  
            "[addr]<20>"  
        ]  
    },  
    "LDV": {  
        "arguments": ["[addr]"],  
        "opcode": "0001",  
        "execution": [  
            ["SAR", "[addr]", "r", ["<mem_addr>", "SAR"]],  
            ["X", "SDR", "r"],  
            ["", "", "r", ["ADD", "Z", "X", "Y", ""],  
            ["AKKU", "Z", ""]]  
        ]  
    }  
}
```

```
[ "", "", "r" ] ,  
[ "", "", "r" , [ "SDR" , "<mem_data>" ] ],  
[ "AKKU" , "SDR" , "" ]  
],  
"translation": [  
    "0001",  
    "[addr]<20>"  
]  

```

```
"arguments": ["[addr]"],
"opcode": "0101",
"execution": [
    ["SAR", "[addr]", "r", ["<mem_addr>", "SAR"]],
    ["X", "AKKU", "r"],
    ["", "", "r", ["SDR", "<mem_data>"]],
    ["Y", "SDR", ""],
    ["OR", "Z", "X", "Y", ""],
    ["AKKU", "Z", ""]
],
"translation": [
    "0101",
    "[addr]<20>"
]
},
"XOR": {
    "arguments": ["[addr]"],
    "opcode": "0110",
    "execution": [
        ["SAR", "[addr]", "r", ["<mem_addr>", "SAR"]],
        ["X", "AKKU", "r"],
        ["", "", "r", ["SDR", "<mem_data>"]],
        ["Y", "SDR", ""],
        ["XOR", "Z", "X", "Y", ""],
        ["AKKU", "Z", ""]
],
"translation": [
    "0110",
    "[addr]<20>"
]
},
"HALT": {
    "arguments": [],
    "opcode": "11110000",
    "execution": [
        ["PAUSE"]
],
"translation": [
    "11110000",

```

```
        "00000000000000000000"
    ],
},
"NOT": {
    "arguments": [],
    "opcode": "11110001",
    "execution": [
        ["X", "AKKU", ""],
        ["NEG", "Z", "X", "", ""],
        ["AKKU", "Z", ""]
    ],
    "translation": [
        "11110001",
        "00000000000000000000"
    ]
},
"RAR": {
    "arguments": [],
    "opcode": "11110010",
    "execution": [
        ["X", "AKKU", ""],
        ["RR", "Z", "X", "", ""],
        ["AKKU", "Z", ""]
    ],
    "translation": [
        "11110010",
        "00000000000000000000"
    ]
},
"EQL [addr)": {
    "arguments": ["[addr]"],
    "opcode": "0111",
    "execution": [
        ["SAR", "[addr]", "r", [<mem_addr>, "SAR"]],
        ["X", "AKKU", "r"],
        ["", "", "r", ["SDR", "<mem_data>"]],
        ["Y", "SDR", ""],
        ["EQ", "Z", "X", "Y", ""],
        ["AKKU", "Z"]
    ]
}
```

```
        ],
        "translation": [
            "0111",
            "[addr]<20>"
        ]
    },
    "JMP": {
        "arguments": "[addr]",
        "opcode": "1000",
        "execution": [
            ["IAR", "[addr]", ""]
        ],
        "translation": [
            "1000",
            "[addr]<20>"
        ]
    },
    "JMN": {
        "arguments": "[addr]",
        "opcode": "1001",
        "execution": [
            ["X", "AKKU", ""],
            ["Y", "[addr]", ""],
            ["IF", ["X", "0", "<"], ["PC", "[addr]"], "", ""]
        ],
        "translation": [
            "1001",
            "[addr]<20>"
        ]
    }
}
```