Design Document: NEUTERO (Neural Network based Image Classification/Detection on Heterogeneous Platforms)

Oguz Mutlu
Yavuz Karaca
Muhammed Rodi Düger
Junchi Li
Umut Sezen

Supervisors:
Christopher Münch
Soyed Tuhin Ahmed

ITEC Tahoori
Chair of Dependable Nano Computing

June 30, 2022

Contents

1 Introduction

With this document we are presenting the architecture of Neutero, which is based on the Model-View-Controller design pattern. In the following each class will be precisely described, connections within the system will be presented and chronological events of various typical processes will be explained through Sequence diagrams.

2 System Structure

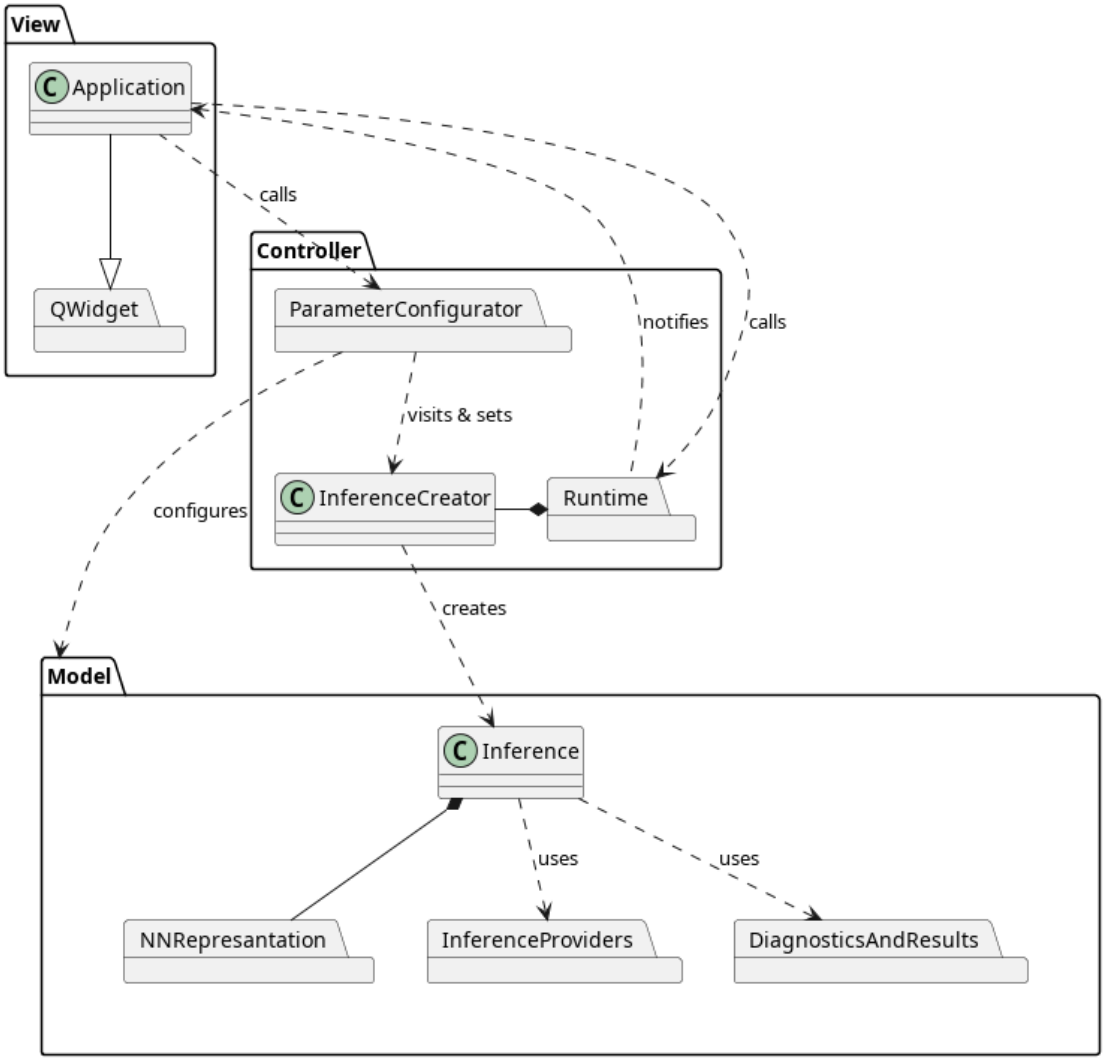
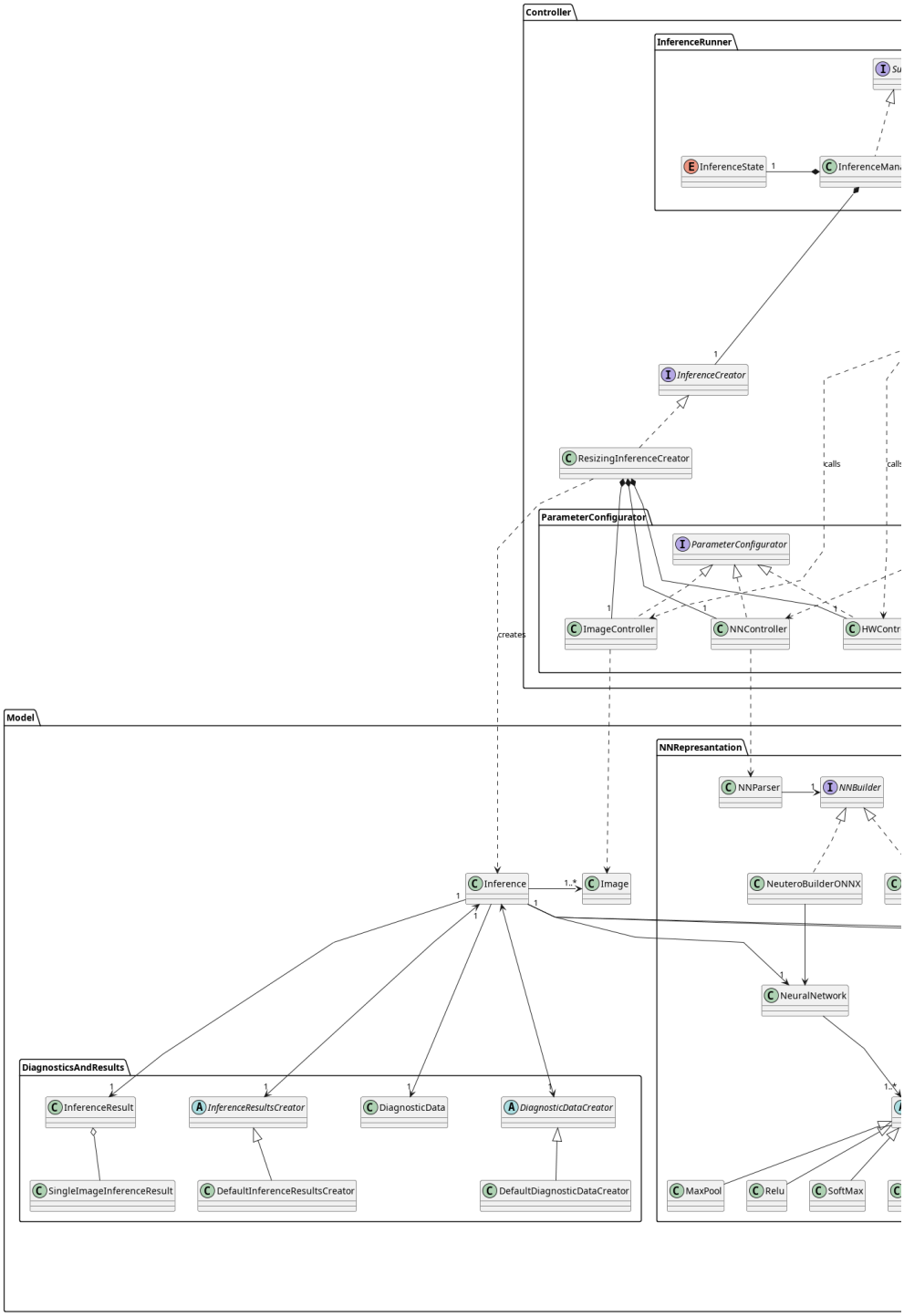


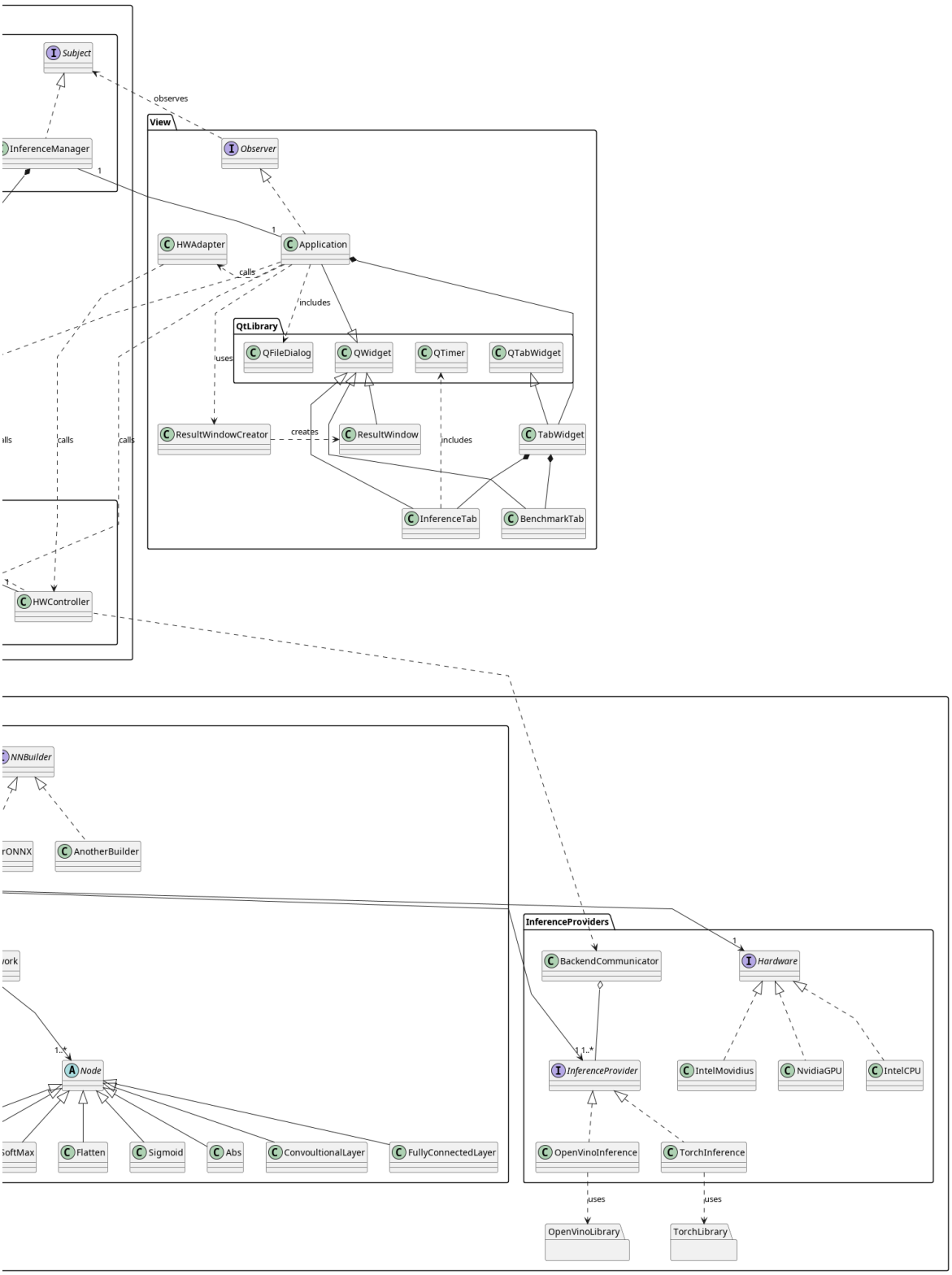
Figure 1: Structural Overview

Our system utilizes the "Model-View-Controller" architecture. If needed, the "Model" part and the "View-Controller" part of the design can be modified without modifying another, as long as the method calls and the general structure of the program is in consideration. "Model" part is mainly responsible for the data structures and the communication between the different libraries used for the inferencing. "View" part is responsible for communicating with the user through a Graphical User Interface. "Controller" part is responsible for making the communication between "View" and "Model", and also modifying/checking the program in runtime.

3 Class Descriptions

3.1 Overview




Subject
observes
InferenceManager
1
View
Observer
1
HWAdapter
Application
calls
includes
QtLibrary
QFileDialog
QWidget
QTimer
QTabWidget
uses
ResultWindowCreator
creates
ResultWindow
includes
TabWidget
InferenceTab
BenchmarkTab
calls
calls
calls
HWController
NNBuilder
AnotherBuilder
1..*
Node
Flatten
Sigmoid
Abs
ConvoultionalLayer
FullyConnectedLayer
InferenceProviders
BackendCommunicator
1
Hardware
1 1..*
InferenceProvider
IntelMovidius
NvidiaGPU
IntelCPU
OpenVinoInference
TorchInference
uses
uses
OpenVinoLibrary
TorchLibrary

3.2 Model

The model package contains the logic of running an inference. It contains representations of entities related to inferencing.

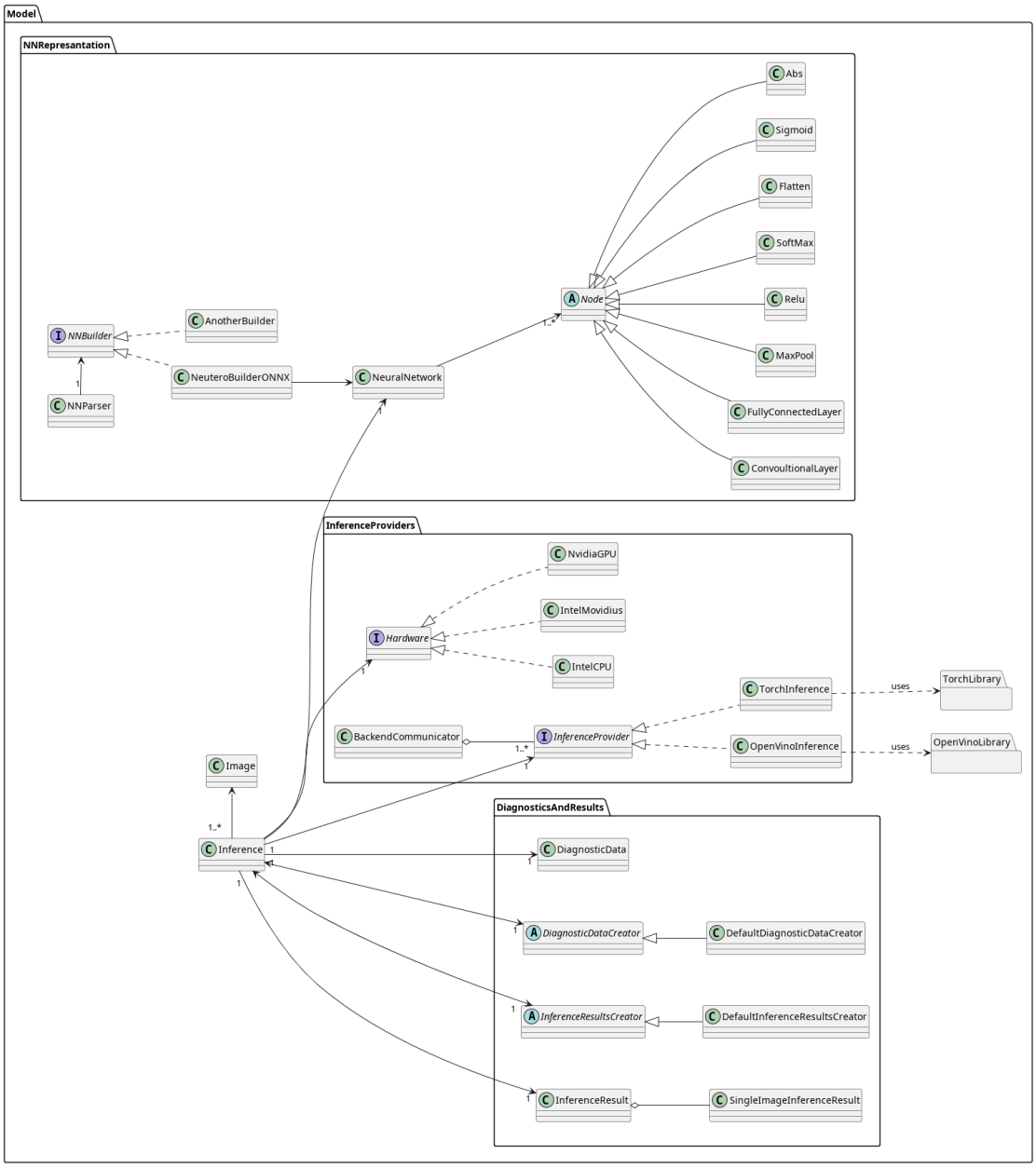
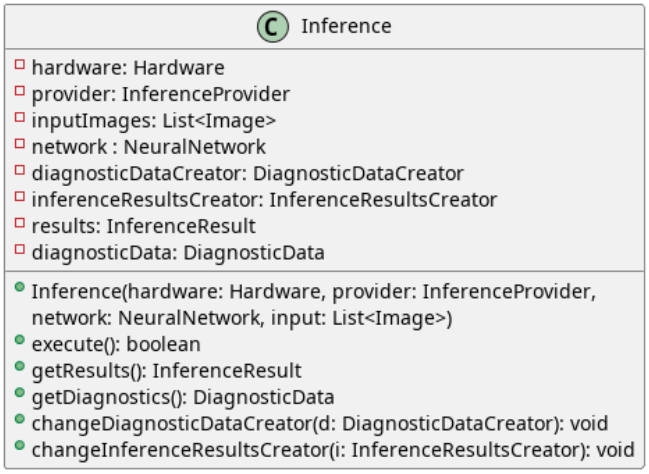


Figure 2: Simplified Class Diagram of Model

3.2.1 Inference class and Image class

Contains representations of an inference process, hardware and input images.



3.2.1.1 Inference

Type: Class

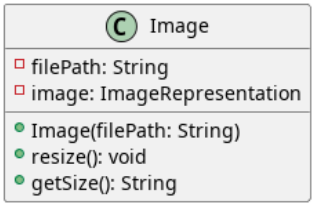
Description: An inference process with exactly one neural network, exactly one hardware component and one or more images.

Attributes

- **hardware: Hardware** Describes the hardware component, the inference runs on
- **provider: InferenceProvider** Describes which inference provider is used
- **inputImages: List<Image>** List of input images
- **network: NeuralNetwork** Represenatation of the neural network for inference
- **diagnosticDataCreator: DiagnosticDataCreator** Used to create diagnostics
- **inferenceResultsCreator: InferenceResultsCreator** Used to create results
- **results: InferenceResult** Describes results of inference
- **diagnosticData: DiagnosticData** Describes diagnostics of inference

Methods

- **Inference(hardware: Hardware, provider: InferenceProvider, network: NeuralNetwork , input: List<Image>)** Constructor of the class, creates an inference process with given arguments
- **execute(): boolean** Executes the already set up inference using the corresponding inference provider, moreover saves results and diagnostics to corresponding attributes, returns True when complete
- **getResults(): InferenceResult** Get results of inference
- **getDiagnostics(): DiagnosticData** Get diagnostic data of inference
- **changeDiagnosticDataCreator(d: DiagnosticDataCreator): void** Change the diagnostic data creator, affects how diagnostic data is created
- **changeInferenceResultsCreator(i: InferenceResultsCreator): void** Change the inference result creator, affects how inference results are created



3.2.1.2 Image

Type: Class

Description: Represents a single image file

Attributes

- **filePath: String** File path of the image
- **image: ImageRepresentation** Representation of the image (for example C++ library)

Methods

- **Image(filePath: String)** Constructor of the class

- **resize(width: Integer, height: Integer): void** Resizes the image to the specified size
- **getSize(): String** Returns the current size of the image

3.2.2 Internal neural network representation

Contains classes related to the internal representation of a neural network topology

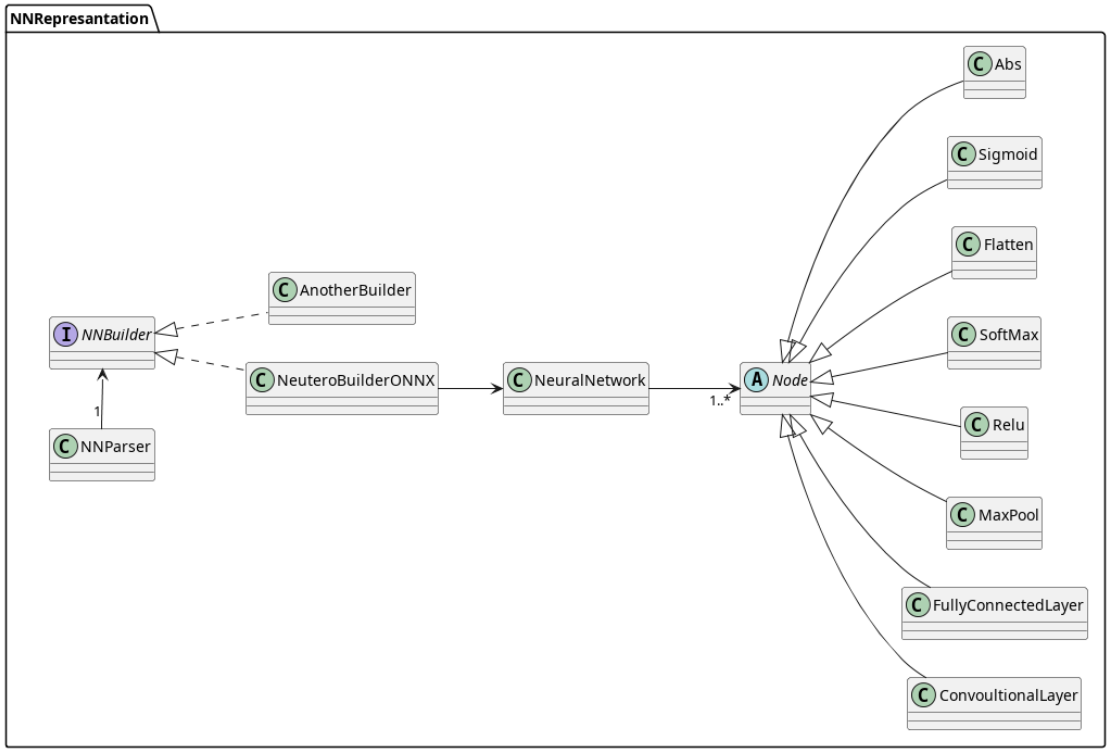
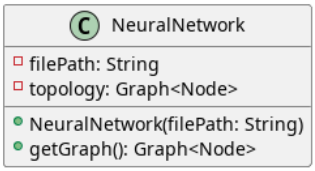


Figure 3: Simplified Class Diagram of package NNRepresentation



3.2.2.1 NeuralNetwork

Type: Class

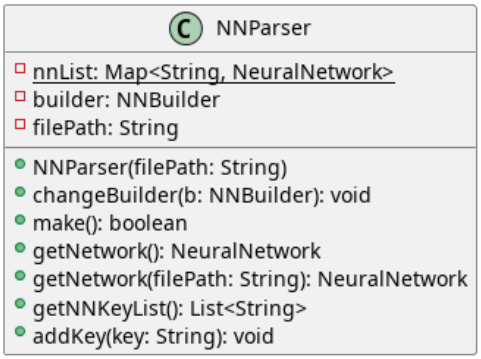
Description: Represents a neural network in runtime

Attributes

- **filePath: String** File path of the corresponding ONNX file
- **topology: Graph<Node>** Graph representation of the topology of the neural network

Methods

- **NeuralNetwork(filePath: String)** Constructor of the class
- **getGraph(): Graph<Node>** Gets the graph representation of the neural network topology.



3.2.2.2 NNParser

Type: Class

Description: Director class for an *NNBuilder*, creates the internal representation from a given neural network file path (e.g. ONNX file).

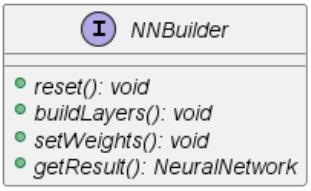
Attributes

- **nnList: Map<String, NeuralNetwork>** A static map, which maps file paths to their converted *NeuralNetwork* objects, works as a cache
- **builder: NNBuilder** Used for converting neural network file
- **filePath: String** File path of the neural network file which is to be converted

Methods

- **NNParser(filePath: String)** Constructor of the class

- **changeBuilder(b: NNBuilder): void** Changes the *NNBuilder*, for example a different *NNBuilder* may be used for converting from another neural network file type
- **make(): boolean** Executes the conversion and gives the converted *NeuralNetwork* object obtained from *NNBuilder* to the map, returns True when complete
- **getNetwork(): NeuralNetwork** Get the converted neural network from file at *filePath* attribute
- **getNetwork(filePath: String): NeuralNetwork** Get the converted neural network from file at *filePath* parameter
- **getNNKeyList(): List<String>** Get a list of file paths from the *nnList* map
- **addKey(key: String): void** Add a key to the *nnList* map



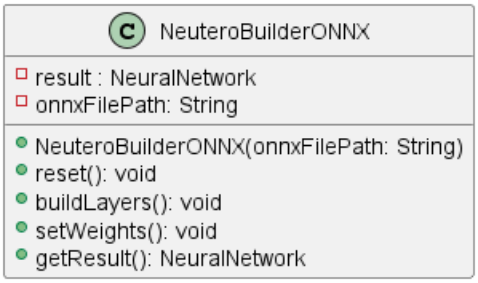
3.2.2.3 NNBuilder

Type: Interface

Description: Builder for *NeuralNetwork* out of a neural network file, which is specified in classes implementing this interface

Methods

- **reset(): void** Resets the saved *NeuralNetwork* object
- **buildLayers(): void** Builds the layers of the *NeuralNetwork* object
- **setWeights(): void** Builds the layers of the *NeuralNetwork* object
- **getResult(): NeuralNetwork** Get the created *NeuralNetwork* object



3.2.2.4 NeuteroBuilderONNX implements *NNBuilder*

Type: Class

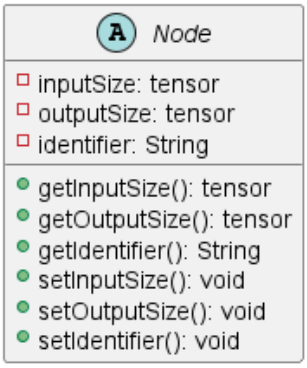
Description: Builder for *NeuralNetwork* out of an ONNX file, that is methods are implemented specifically for converting ONNX file

Attributes

- **result : NeuralNetwork** Created *NeuralNetwork* object
- **onnxFilePath: String** Path of the ONNX file

Methods

- NeuteroBuilderONNX(onnxFilePath: String) Constructor of the class
- Methods specified in *NNBuilder* interface



3.2.2.5 Node

Type: Abstract class

Description: Represents a node in a graph represenatation of a neural network, for example a layer, an activation function and an mathematical operation are all nodes

Attributes

- **inputSize: tensor** Input size of the node, "tensor" is a placeholder for a tensor type in an external math library
- **outputSize: tensor** Output size of the node
- **identifier: String** Identifier of the node

Methods

- **getInputSize(): tensor** Get the input size
- **getOutputSize(): tensor** Get the output size
- **getIdentifier(): String** Get the identifier
- **setInputSize(): void** Set the input size
- **setOutputSize(): void** Set the output size
- **setIdentifier(): void** Set the identifier

3.2.2.6 ConvolutionalLayer extends *Node*

Type: Class

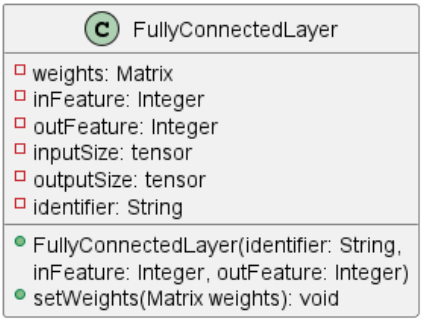
Description: Represents a convolutional layer

Attributes

- **weights: List<Matrix>** Weights of the layer
- **inChannel : Integer**
- **outChannel: Integer**
- **kernelSize: Integer**
- **padding : Integer**
- **stride: Integer**
- And attributes specified in parent class

Methods

- **ConvoultionalLayer(identifier: String, inChannel: Integer, outChannel: Integer, kernelSize: Integer, padding: Integer, stride: Integer)** Constructor of the class
- **setWeights(weights: List<Matrix>): void** Set the weigths
- And methods specified in parent class



3.2.2.7 FullyConnectedLayer extends *Node*

Type: Class

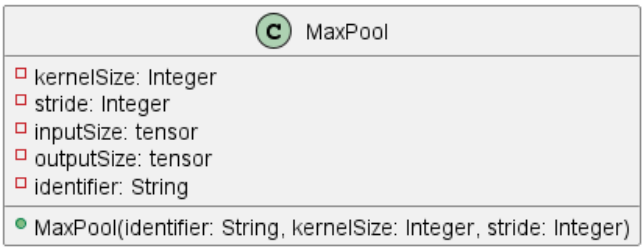
Description: Represents a fully connected layer

Attributes

- **weights: Matrix** Weights of the layer
- **inFeature: Integer**
- **outFeature: Integer**
- And attributes specified in parent class

Methods

- **FullyConnectedLayer(identifier: String, inFeature: Integer, outFeature: Integer)** Constructor of the class
- **setWeights(weights: Matrix): void** Set the weigths
- And methods specified in parent class



3.2.2.8 MaxPool extends *Node*

Type: Class

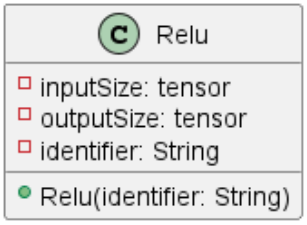
Description: Represents a max pool layer

Attributes

- **kernelSize: Integer**
- **stride: Integer**
- And attributes specified in parent class

Methods

- **MaxPool(identifier: String, kernelSize: Integer, stride: Integer)** Constructor of the class
- And methods specified in parent class



3.2.2.9 Relu extends *Node*

Type: Class

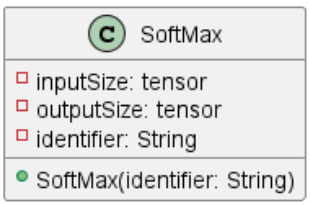
Description: Represent the activation function Relu

Attributes

- Specified in parent class

Methods

- **Relu(identifier: String)** Constructor of the class
- And methods specified in parent class



3.2.2.10 SoftMax extends *Node*

Type: Class

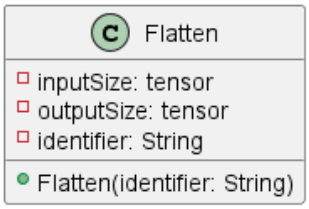
Description: Represents the activation function Soft Max

Attributes

- Specified in parent class

Methods

- **SoftMax(identifier: String)** Constructor of the class
- And methods specified in parent class



3.2.2.11 Flatten extends *Node*

Type: Class

Description: Represents a flatten operation

Attributes

- Specified in parent class

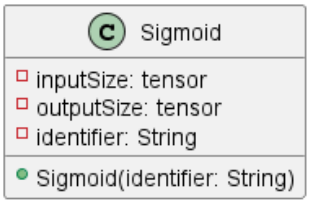
Methods

- **Flatten(identifier: String)** Constructor of the class
- And methods specified in parent class

3.2.2.12 Sigmoid extends *Node*

Type: Represents the activation function Sigmoid

Description:

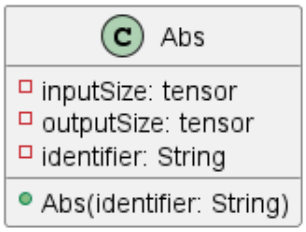


Attributes

- Specified in parent class

Methods

- **Sigmoid(identifier: String)** Constructor of the class
- And methods specified in parent class



3.2.2.13 Abs extends *Node*

Type: Class

Description: Represents the operation "absolute value"

Attributes

- Specified in parent class

Methods

- **Abs(identifier: String)** Constructor of the class
- And methods specified in parent class

3.2.3 Inference Providers

Contains representations of various inference providers and hardware components.

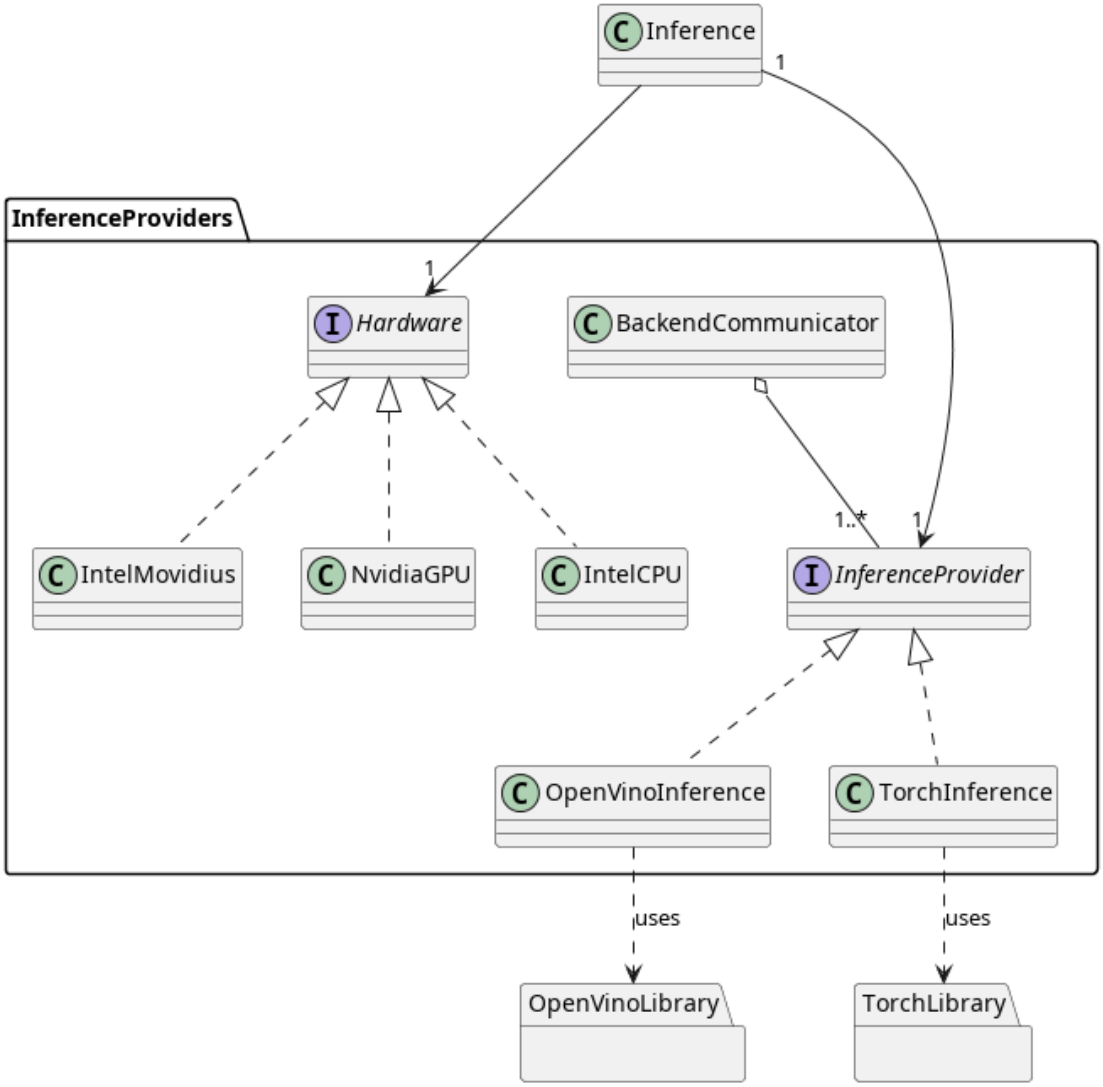
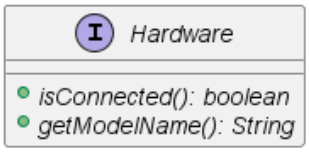


Figure 4: Simplified Class Diagram of package InferenceProviders including *Inference* Class and external libraries



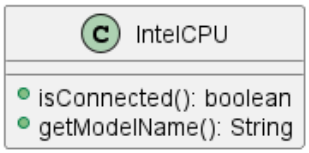
3.2.3.1 Hardware

Type: Interface

Description: Describes a hardware component

Methods

- **isConnected(): boolean** Checks whether the hardware, represented by the class, is connected, using *BackendCommunicator* class
- **getModelName(): String** Returns the model name of the hardware, using *BackendCommunicator* class



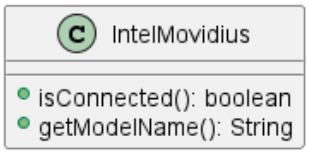
3.2.3.2 IntelCPU implements Hardware

Type: Class

Description: Represents an Intel CPU

Methods

This class implements the methods of the *Hardware* interface



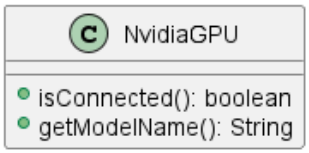
3.2.3.3 IntelMovidius implements Hardware

Type: Class

Description: Represent an Intel Movidius VPU

Methods

This class implements the methods of the *Hardware* interface



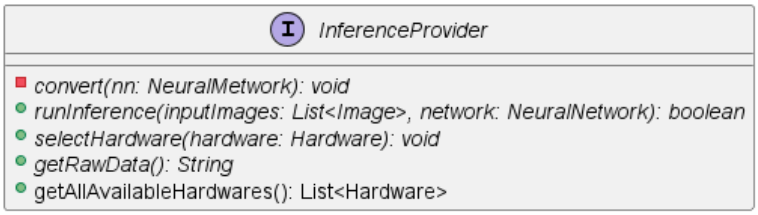
3.2.3.4 NvidiaGPU implements Hardware

Type: Class

Description: Represents an Nvidia GPU

Methods

This class implements the methods of the *Hardware* interface



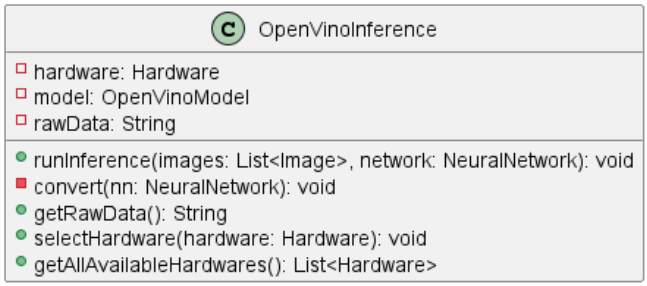
3.2.3.5 InferenceProvider

Type: Interface

Description: Represents an inference provider and makes it possible to access all inference libraries uniformly

Methods

- **convert(nn: NeuralNetwork): void** Converts the NeuralNetwork instance given as parameter to library specific format
- **runInference(inputImages: List<Image>, nn: NeuralNetwork): boolean** Converts the NeuralNetwork instance given as parameter into library specific format with the *convert()* method and runs inference on it with the given list of images
- **selectHardware(hardware: Hardware): void** Selects the hardware for the inference
- **getRawData(): String** Returns the inference result as raw data
- **getAllAvailableHardware(): List<Hardware>** Returns a list of all available hardwares, which are in scope of the specific library. For example Intel Movidius and Intel CPU for OpenVINO



3.2.3.6 OpenVinoInference implements *InferenceProvider*

Type: Class

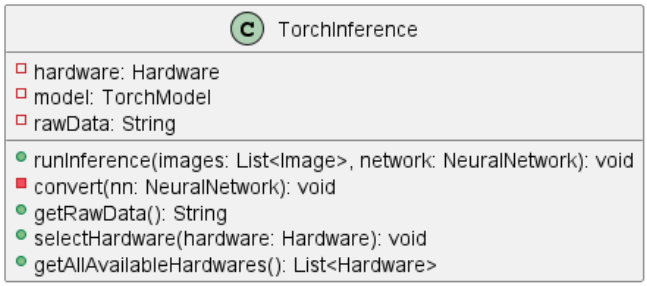
Description: This class runs the actual inference with using OpenVINO library

Attributes

- **hardware: Hardware** Selected hardware, which the inference will run on
- **model: OpenVinoModel** OpenVINO-specific representation of the neural network, converted from *NeuralNetwork* representation through *convert()* method

Methods

This class implements the methods of the *InferenceProvider* interface



3.2.3.7 TorchInference implements *InferenceProvider*

Type: Class

Description: This class runs the actual inference with using LibTorch library

Attributes

- **hardware: Hardware** Selected hardware,which the inference will run on
- **model: TorchModel** Torch-specific representation of the neural network, converted from *NeuralNetwork* representation through *convert()* method

Methods

This class implements the methods of the *InferenceProvider* interface

3.2.3.8 BackendCommunicator

Type: Class

Description: This class makes it possible to communicate with hardware through inference libraries to read information about available hardware

Attributes

- **singletonInstance: BackendCommunicator** An attribute to realize Singleton design pattern
- **providers: List<InferenceProvider>** List of instances of each inference provider

Methods

- **BackendCommunicator()** Private constructor
- **getBackendCommunicator(): BackendCommunicator** Static getter for the singleton instance
- **getAvailableHardwares(): Map<Hardware, List<InferenceProvider»** Get a list of all available hardware with corresponding inference providers

3.2.4 Diagnostics and Results

Contains classes related to inference results and diagnostic data.

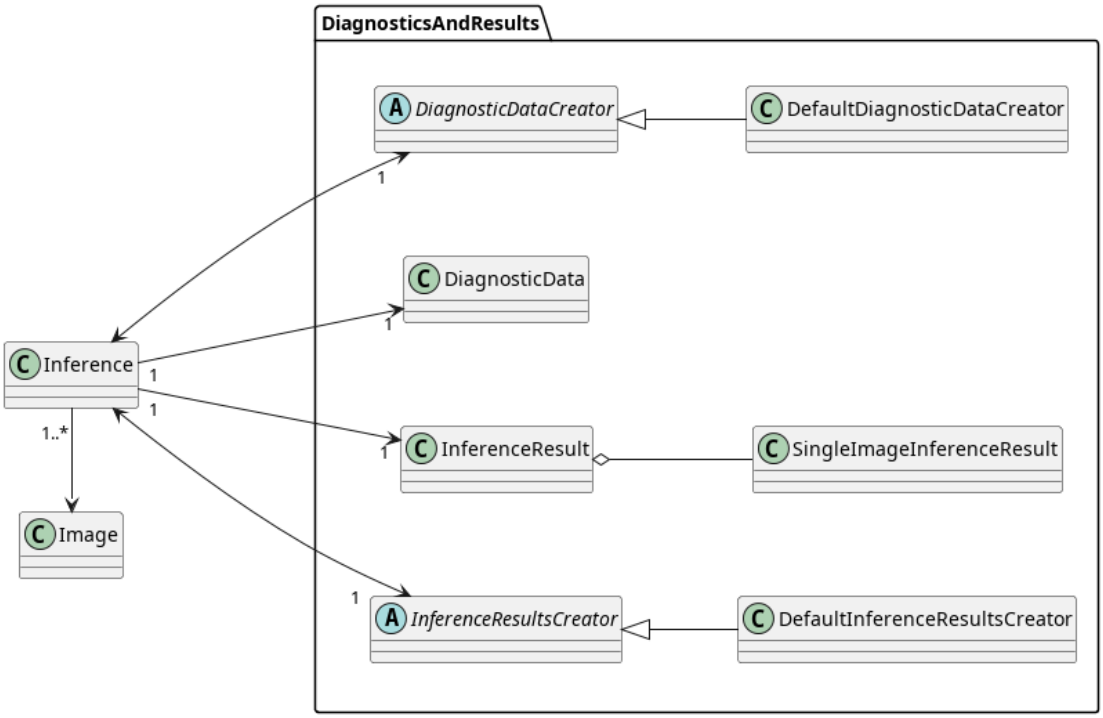
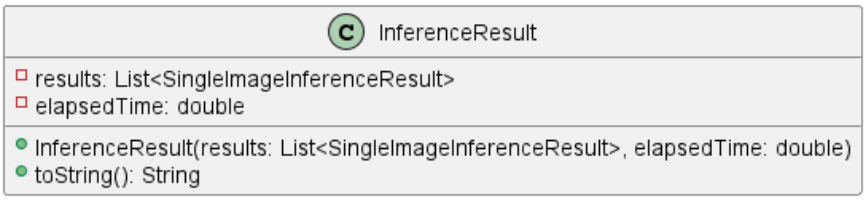


Figure 5: Simplified Class Diagram of package *Diagnostics and Results* including *Inference* and *Image* classes



3.2.4.1 InferenceResult

Description: Represents a result of an inference process, which is represented by *Inference* class.

Type: Class

Attributes

- **results: List<SingleImageInferenceResult>** List of results of single images
- **elapsedTime: double** Elapsed time for the whole inference process

Methods

- **InferenceResult(results: List<SingleImageInferenceResult>, elapsedTime: double)** Constructor of the class
- **toString(): String** Calls *toString* methods of the objects from list and concatenates all of them

3.2.4.2 SingleImageInferenceResult

Description: Represents result of an inference applied on exactly one image

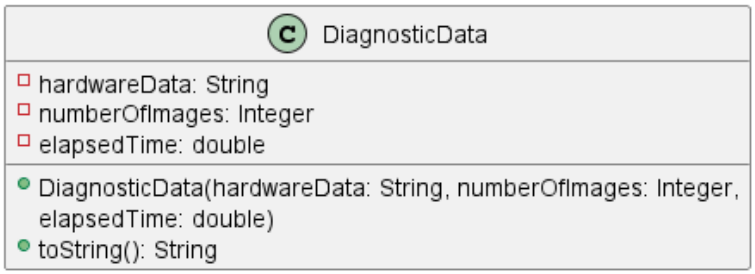
Type: Class

Attributes

- **image: Image** Corresponding image object
- **confidences: Map<String, String>** Mapping of classification labels to confidence levels

Methods

- **SingleImageInferenceResult(image: Image, confidences: Map<String, String>)** Constructor of the class
- **toString(): String** Converts the name of the image together with the confidence map to String



3.2.4.3 DiagnosticData

Description: Represents diagnostic data of an inference process, which is represented by *Inference* class.

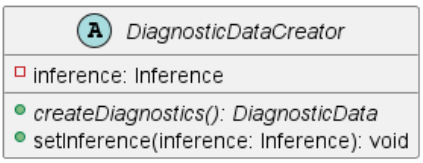
Type: Class

Attributes

- **hardwareData: String** Model data obtained from hardware
- **numberOfImages: Integer** Number of images processed in the inference
- **elapsedTime: double** Elapsed time for the whole inference process

Methods

- **DiagnosticData(hardwareData: String, numberOfImages: Integer, elapsedTime: double)** Constructor of the class
- **toString(): String** Converts the data stored in attributes to String and concatenates them



3.2.4.4 DiagnosticDataCreator

Description: Describes how diagnostic data is created

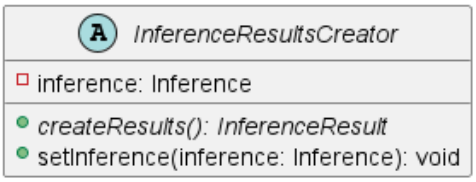
Type: Abstract class

Attributes

- **inference: Inference** Corresponding inference process

Methods

- **createDiagnostics(): DiagnosticData** Creates diagnostic data with information from the *inference* and returns it
- **setInference(inference: Inference): void** Set the *inference* process



3.2.4.5 InferenceResultsCreator

Description: Describes how inference results are created

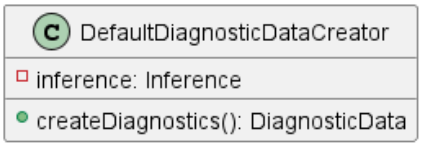
Type: Abstract class

Attributes

- **inference: Inference** Corresponding inference process

Methods

- **createResults(): InferenceResult** Creates diagnostic data with information from the *inference* and returns it
- **setInference(inference: Inference): void** Set the *inference* process



3.2.4.6 DefaultDiagnosticDataCreator extends *DiagnosticDataCreator*

Description: Default Neutero diagnostic data creator, fills out the information in *DiagnosticData* class

Type: Class

Attributes Described in parent class

Methods Described in parent class

3.2.4.7 DefaultInferenceResultsCreator extends *InferenceResultsCreator*

Description: Default Neutero inference results creator, fills out the information in *InferenceResult* class

Type: Class

Attributes Described in parent class

Methods Described in parent class

3.3 View

The View package contains the front-end related functionality and the graphical user interface design based on Qt library.

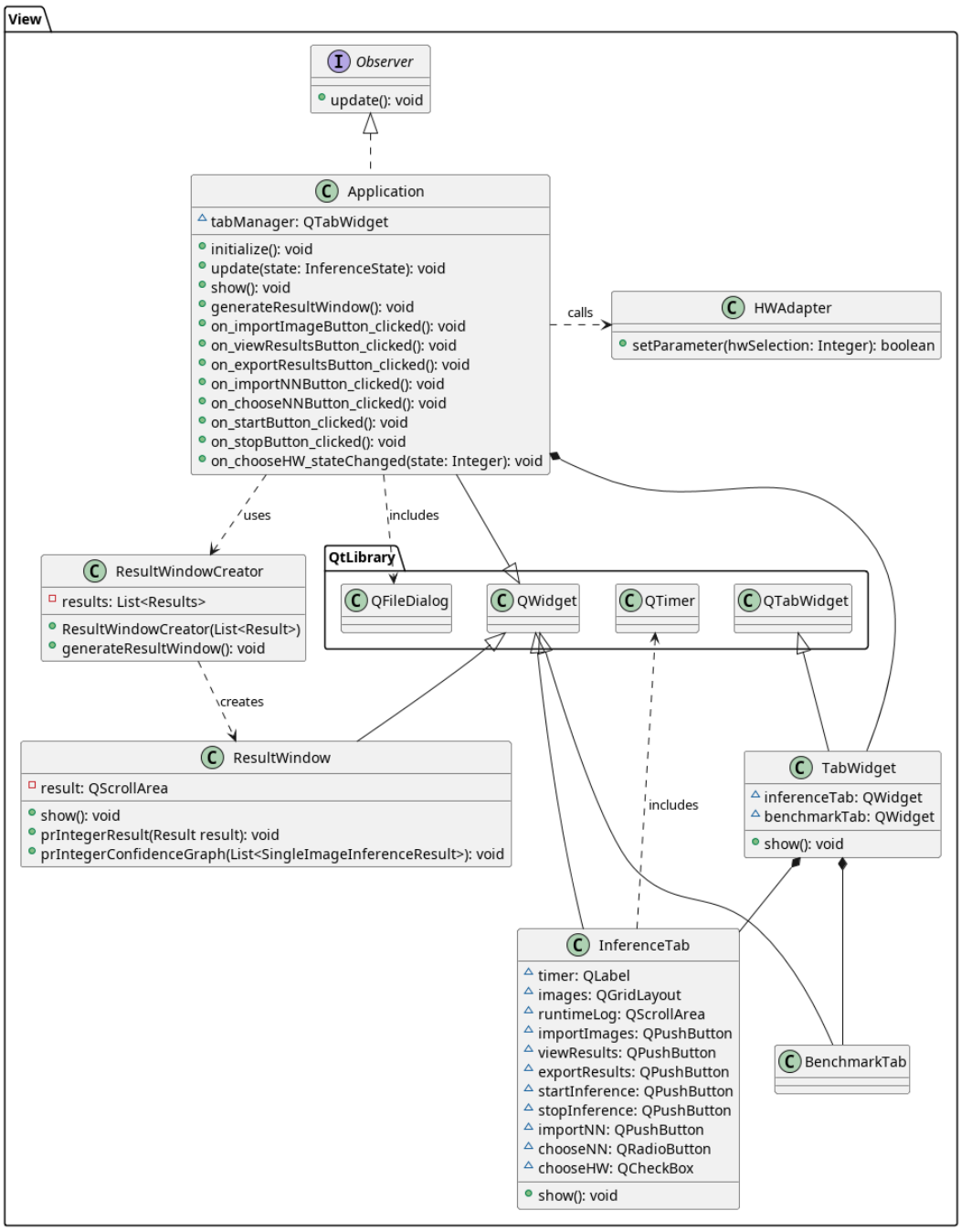
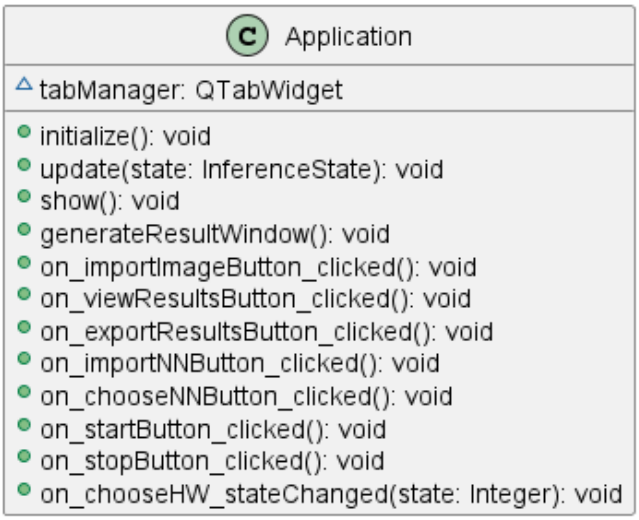


Figure 6: Class Diagram of View

3.3.1 **Application** extends *QWidget* implements *Observer*



Type: Class

Description: This class represents the main application window and provides an interface for user interaction

Attributes

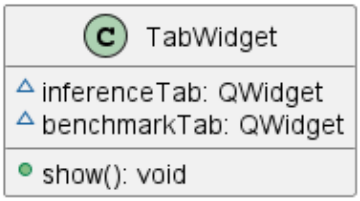
- **tabManager: QTabWidget** manages tab layout

Methods

- **initialize(): void** initializes the application window and it's child widgets and displays them all
- **update(state: InferenceState): void** update application display
- **show(): void** displays the application window
- **generateResultWindow(): void** calls ResultWindowCreator to generate results
- **onImportImageButtonClicked(): void** defines action of the import image button
- **onViewResultsButtonClicked(): void** defines action of the view result button
- **onExportResultsButtonClicked(): void** defines action of the export results button

- **onImportNNButtonClicked(): void** defines action of the import nn button
- **onChooseNNButtonClicked(): void** defines action of the choose nn button
- **onStartButtonClicked(): void** defines action of the import start button
- **onStopButtonClicked(): void** defines action of the import stop button
- **onChooseHWStateChanged(state: Integer): void** defines action of the choose hw button

3.3.2 TabWidget extends QTabWidget



Type: Class

Description: This class defines the layout for InferenceTab and BenchmarkTab

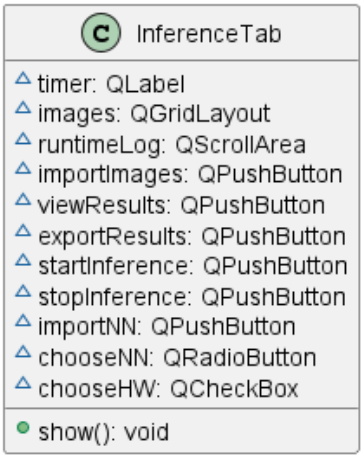
Attributes

- **inferenceTab: QWidget** first tab
- **benchmarkTab: QWidget** second tab

Methods

- **show(): void** displays the layout

3.3.3 InferenceTab extends *QWidget*



Type: Class

Description: is responsible for the layout of the Inference Tab.

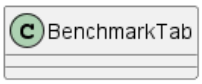
Attributes

- **timer: QLabel** shows elapsed time
- **images: QGridLayout** previews imported images
- **runtimeLog: QScrollArea** shows runtime-log
- **importImages: QPushButton** button for importing images
- **viewResults: QPushButton** button for opening pop-up window showing results
- **exportResults: QPushButton** button for exporting results
- **startInference: QPushButton** button for starting the inference
- **stopInference: QPushButton** button for aborting inference
- **importNN: QPushButton** button for importing neural network topology
- **chooseNN: QRadioButton** button for choosing a single neural network
- **chooseHW: QCheckBox** button for choosing one or multiple hardware components

Methods

- **show(): void** displays the layout

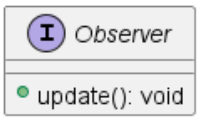
3.3.4 BenchmarkTab extends QWidget



Type: Class

Description: This class will be needed in the future when extending benchmark functionality.

3.3.5 Observer



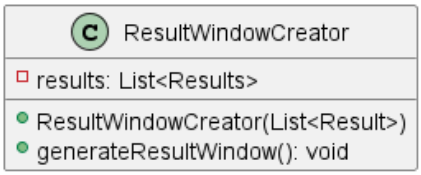
Type: Interface

Description: Observer pattern for application

Methods

- **update(): void** update method that classes have to implement

3.3.6 ResultWindowCreator



Type: Class

Description: creates result windows for each inference

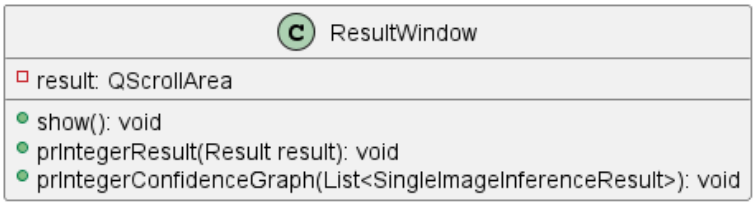
Attributes

- **results: List<InferenceResult>**

Methods

- **ResultWindowCreator(results: List<InferenceResult>): void**
- **generateResultWindow(): void**

3.3.7 ResultWindow



Type: Class

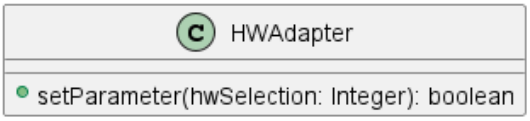
Description: is the graphical representation of confidence results

Attributes

- **result: QScrollArea**

Methods

- **show(): void**
- **printResult(result: InferenceResult): void**
- **printConfidenceGraph(results: List<SingleImageInferenceResult>): void**



HWAdapter

Type: Class

Description: Adapter for translating hardware selection on GUI to HWConfigurator

Methods

- **setParameter(hwSelection: Integer)** calls HWConfigurator with right parameters

3.4 Controller

The Controller package connects the View and Model packages. It accepts user inputs from the View and manipulates the Model accordingly.

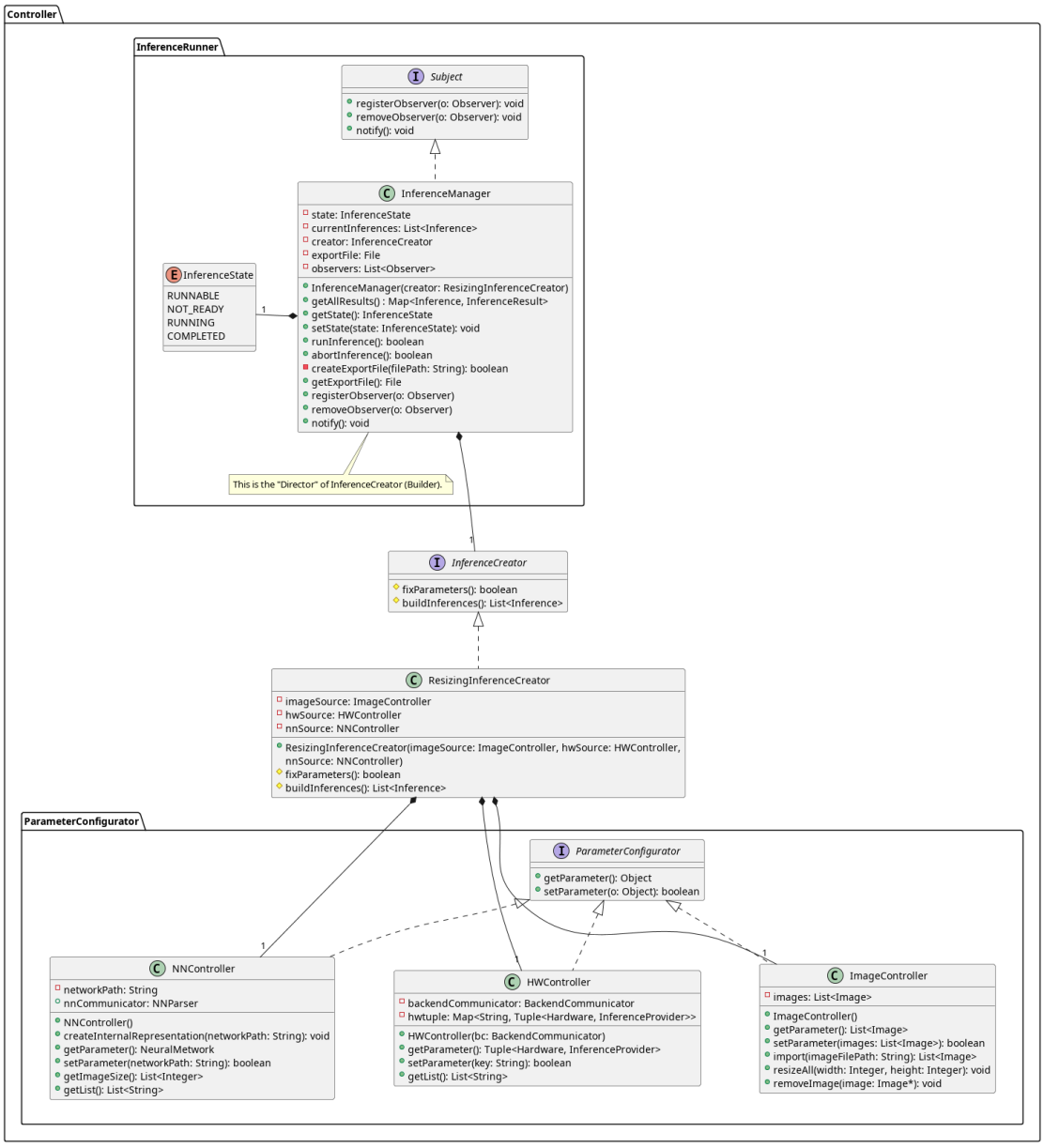
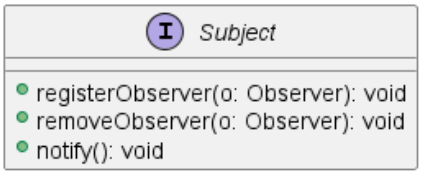


Figure 7: Class Diagram of Controller

3.4.1 InferenceRunner classes

Contains classes that manage the InferenceCreator class with the input from user. In our case, the GUI is responsible for giving these inputs.



3.4.1.1 Subject

Type: Interface

Description: Instances of this class notifies View about changes in Model/Controller. This class utilizes "Observer" design pattern and can be used to implement subjects for observing the state of the program.

Methods

- **registerObserver(o: Observer):void** Registers an observer to an object
- **removeObserver(o: Observer):void** Removes the observer from the observers list of an object
- **notify():void** Notifies the observers of the object

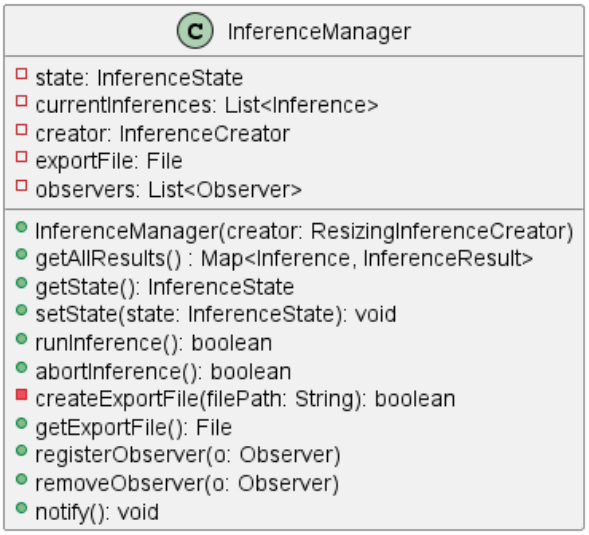
3.4.1.2 InferenceManager implements *Subject*

Type: Class

Description: Director for the InferenceCreator instances. Runs inferences, and tracks their running state for view updates.

Attributes

- **state: InferenceState** Running state of the current instances.
- **currentInferences: List<Inference>** Inferences the program currently handles with.



- **creator: InferenceCreator** Corresponding InferenceCreator to create the inferences.
- **exportFile: File** File to be exported
- **observers: List<Observer>**

Methods

- **InferenceManager(creator: ResizingInferenceCreator)** Constructs an InferenceManager with the given *creator*.
- **getAllResults(): Map<Inference,InferenceResult>** Returns results of all inference instances in currentInferences list.
- **getState(): InferenceState** Gets the current running state.
- **setState(state: InferenceState): void** Sets the current running state. Usually called inside the class, but in cases of rare exceptions this method can be called also outside of the class.
- **runInference(): boolean** Runs the inference. The state would be switched to "running".
- **abortInference(): boolean** Aborts the currently running inference.
- **createExportFile(filePath:String): boolean** A private method which creates the file to be exported when export button in GUI is clicked
- **getExportFile():File** Returns the exportFile attribute. Creates it if it is null.
- This class implements the methods of the *Subject* interface

3.4.1.3 InferenceState

Type: Enum

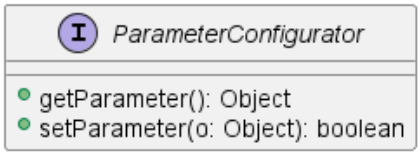
Description: Enum class that represents an inference running state.

Values:

- **RUNNABLE** Parameters are fully initialized and no inference has started yet.
- **NOT_READY** Parameters are not fully initialized.
- **RUNNING** Some inferences are running.
- **COMPLETED** All inference runs are completed and the results are available.

3.4.2 Parameter configurating classes

Contains classes that configure the parameters required for starting inferences.



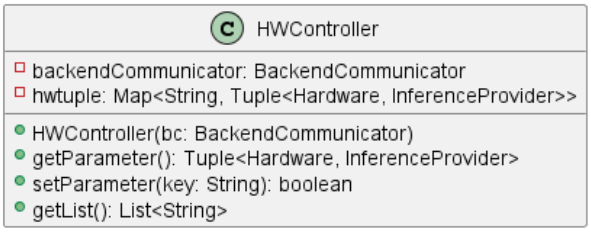
3.4.2.1 ParameterConfigurator

Type: Interface

Description: Interface for generic parameter configuration.

Methods

- **getParameter() : Object** Gets the currently holding parameter. Acquired parameter may be transformed inside the class if necessary.
- **setParameter(o: Object)** Sets the parameter to the provided method argument.



3.4.2.2 HWController implements ParameterConfigurator

Type: Class

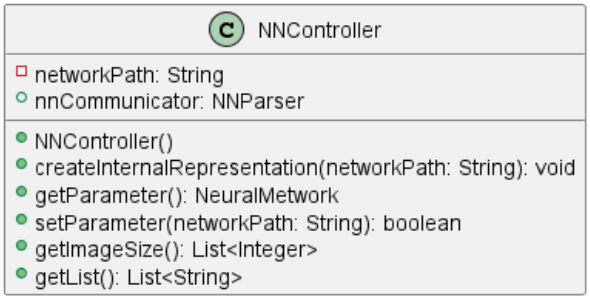
Description: Manages the hardware related parameters.

Attributes

- **backendCommunicator: BackendCommunicator** Corresponding backendCommunicator for hardware information.
- **hwtuple: Map<String, Tuple<Hardware, InferenceProvider»** Currently selected hardware and InferenceProvider tuple(s) with their unique string combination.

Methods

- **HWController(bc: BackendCommunicator)** Constructs a HWController with the given backendCommunicator.
- **getParameter(): Tuple<Hardware, InferenceProvider>** Gets the currently selected hardware/backend tuple(s).
- **setParameter(key: String): boolean** Sets the given parameters about hardware.
- **getList(): List<String>** Gets a complete list of hardware/backend provider tuples from backendCommunicator. Mainly used for GUI.



3.4.2.3 NNController implements ParameterConfigurator

Type: Class

Description: Stores the current selected neural network.

Attributes

- **String networkPath** File path of the currently selected neural network.
- **nnCommunicator: NNParser** NNParser instance for the selected network. NNParser instances are unique for each network.

Methods

- **NNController()** Constructor for this class.
- **createInternalRepresentation(networkPath: String): void** Invokes NNParser's make() method to create the internal representation for the network. Used after the "Run Inference" command.
- **getParameter(): NeuralNetwork** Gets the internal representation of the currently selected neural network.
- **setParameter(networkPath: String): boolean** Sets the file path of the currently selected neural network to the provided argument.
- **getImageSize(): List<Integer>** Gets the supported minimum image size of the network.
- **getList(): List<String>** Gets the list of imported & default networks' file paths. Used for GUI.

3.4.2.4 ImageController implements ParameterConfigurator

Type: Class

Description: Stores all currently imported images. Images may be resized by need.

Attributes

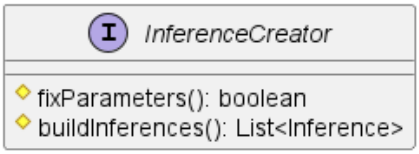
- **images: List<Image>** Currently imported images.

Methods

- **ImageController()** Constructs an ImageController.
- **getParameter(): List<Image>** Gets all currently imported images.
- **setParameter(images: List<Image>): boolean** Sets the selected images as inputs for the inference with the help of import.
- **import(imageFilePath: String): List<Image>** Imports the image(s) from its file path, and converts it to the internal representation of an image. *setParameter()* is invoked after each call.
- **resizeAll(width: Integer, height: Integer): void** Resizes all non-compatible images according to the given width and height.
- **removeImage(image: Image*): void** Removes the given image from the list of imported images. The removed image would no longer be used for inference (unless imported again).

3.4.3 Inference building classes

The interface for creating inferences and their implementations are these classes. They utilize the "Builder" design pattern.



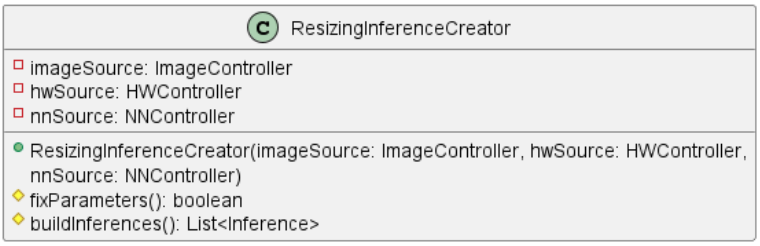
3.4.3.1 InferenceCreator

Type: Interface

Description:

Methods

- **fixParameters(): boolean** Fixes & prepares the parameters for instance creation.
- **buildInferences(): List<Inference>** Creates inference instances to be run by the InferenceManager.



3.4.3.2 ResizingInferenceCreator implements InferenceCreator

Type: Class

Description: Inference creator which can resize images to make them suit to the neural network.

Attributes

- **imageSource: ImageController** Corresponding image parameter class.
- **hwSource: HWController** Corresponding hardware parameter class.
- **nnSource: NNController** Corresponding neural network parameter class.

Methods

- **ResizingInferenceCreator(imageSource: ImageController, hwSource: HWController, nnSource: NNController)** Constructor of the class.
- **fixParameters(): boolean** Checks if all of the parameters are correct. In this class it also resizes any images that are not compliant with the network.
- **buildInferences(): List<Inference>** Creates inference instances by the provided images, hardwares and neural network.

4 Sequence Diagrams

These diagrams are also in the Git repository for better viewing purposes, as `.wsd` files.

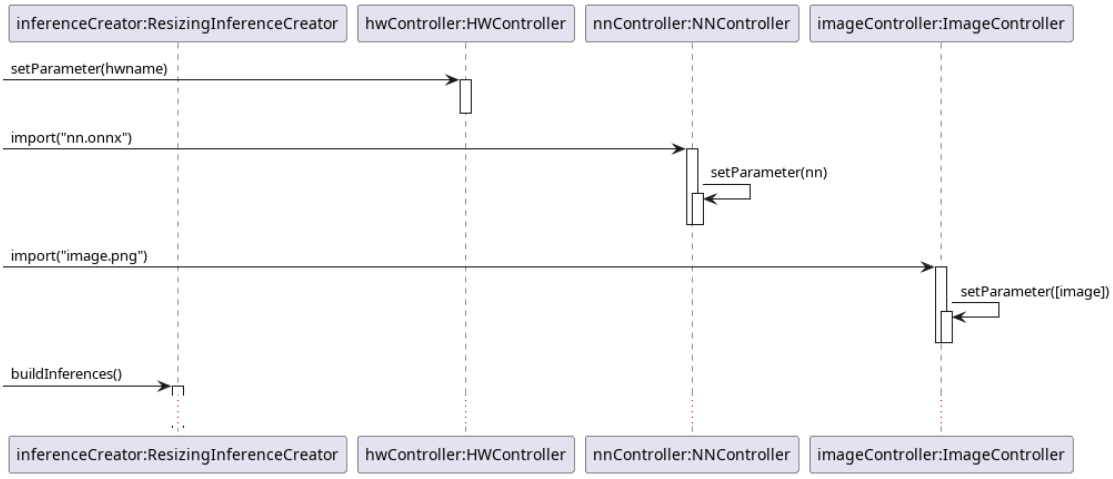


Figure 8: Sequence Diagram for setting parameters inside Controller package.

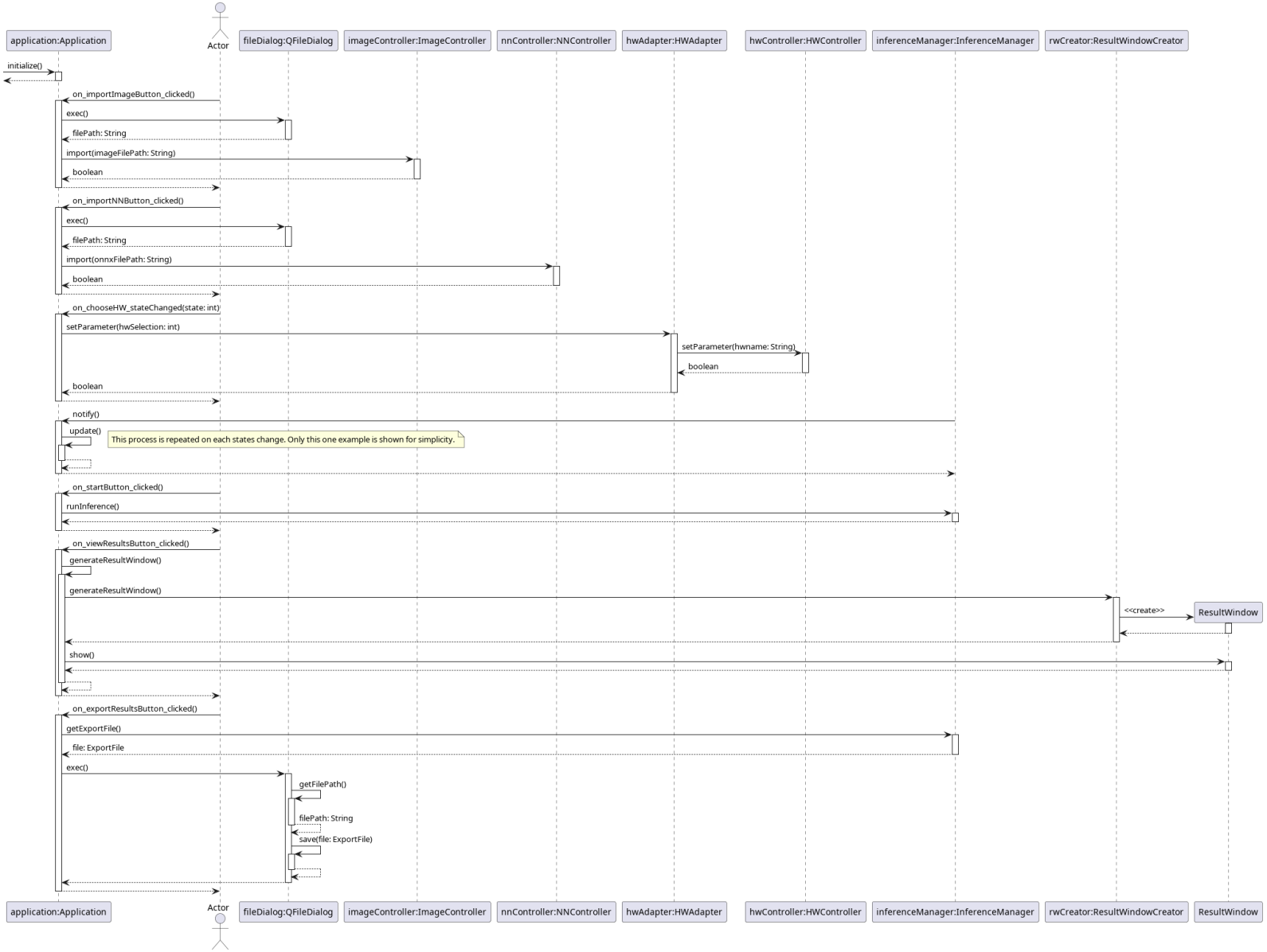


Figure 9: Sequence Diagram for setting parameters inside View package.

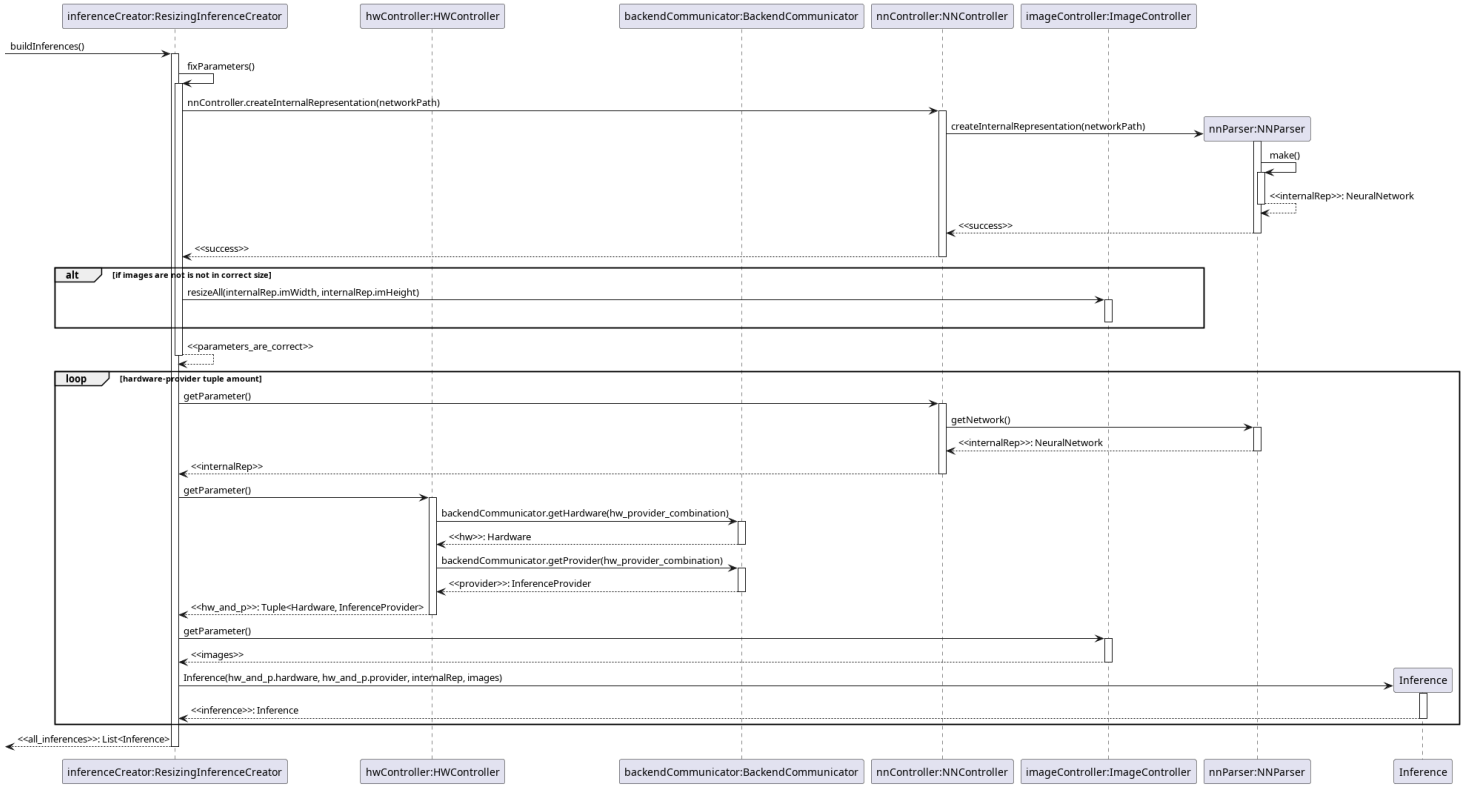


Figure 10: Sequence Diagram for creating *Inference* instances according to the given parameters

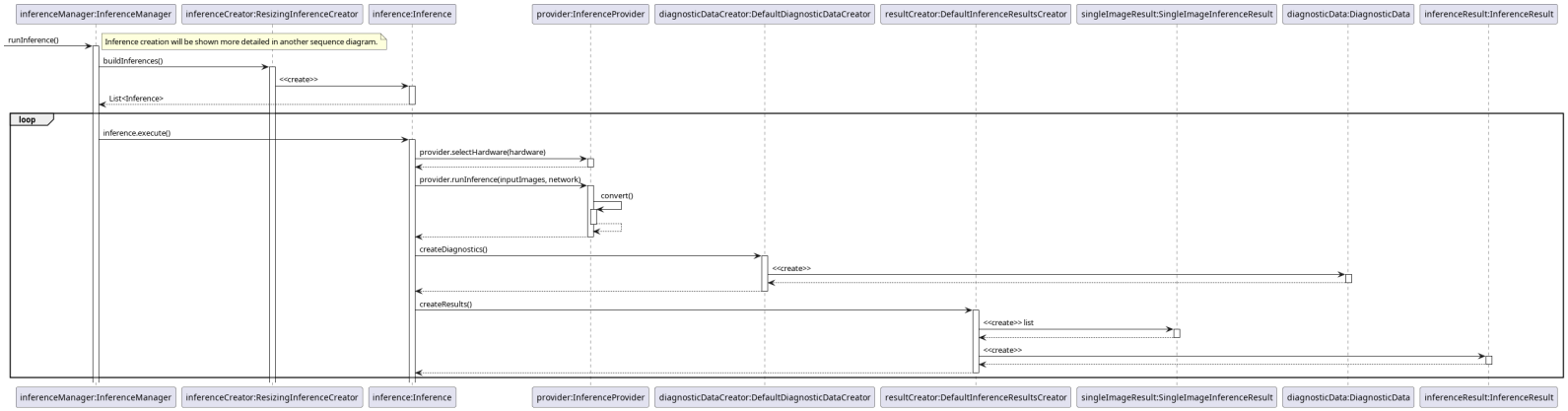


Figure 11: Sequence Diagram for running inferences.