Quick Introduction to

Features

Eventerpretor: Features

- Headless, flow-based programming / Complex Event Processor
- Especially intended to extend home automation software with an independent sensor model and rules engine
- Multithreaded nodes synchronized with *LinkedBlockingQueues*
- Event-based: low CPU usage if there is nothing to process
- Written in Kotlin → All Kotlin / Java libraries can be used
- Stand-alone deployment requires just Java 8 to be installed on the target system

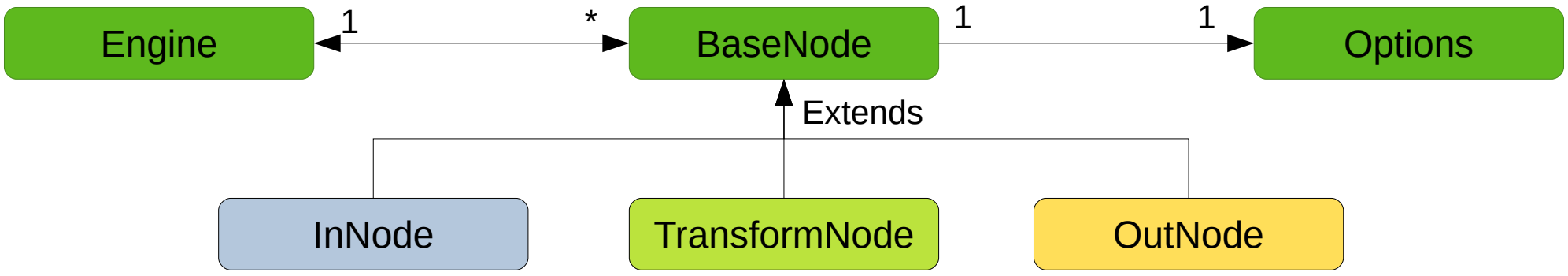
- PostgreSQL database as data storage and for aggregations via trigger / listener
- Secure MQTT / PostgreSQL connections via SSL certificate pinning possible

- Recording and replay of events with identical time gaps for testing
- Websites can directly be integrated for data input and output

- Crossplatform: tested on Windows and Linux - including Raspbian (Raspberry Pi)

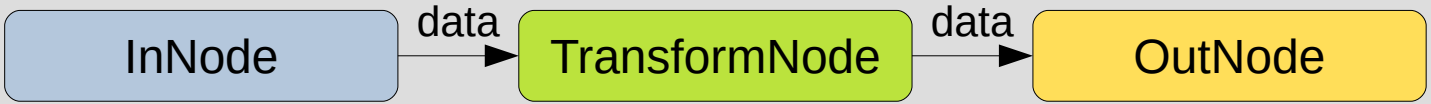
Basic structure

Eventerpretor: Basic structure

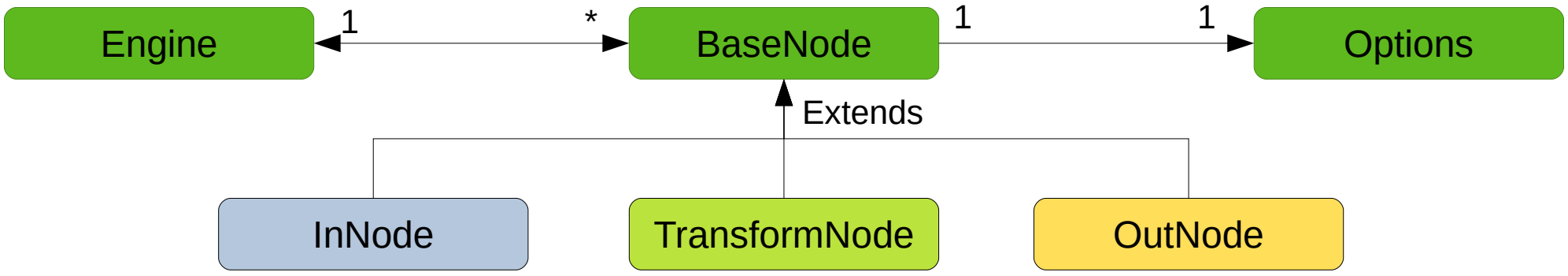


Pipeline

- Pipelines are defined with Groovy scripts which can contain Java code
- Multiple pipelines can be created in one Eventerpretor instance
- Pipelines can be connected with each other
- Pipelines should usually be created as non-cyclic graphs since loops are not prevented!
- The data types of passed data have to be checked / converted by the nodes!



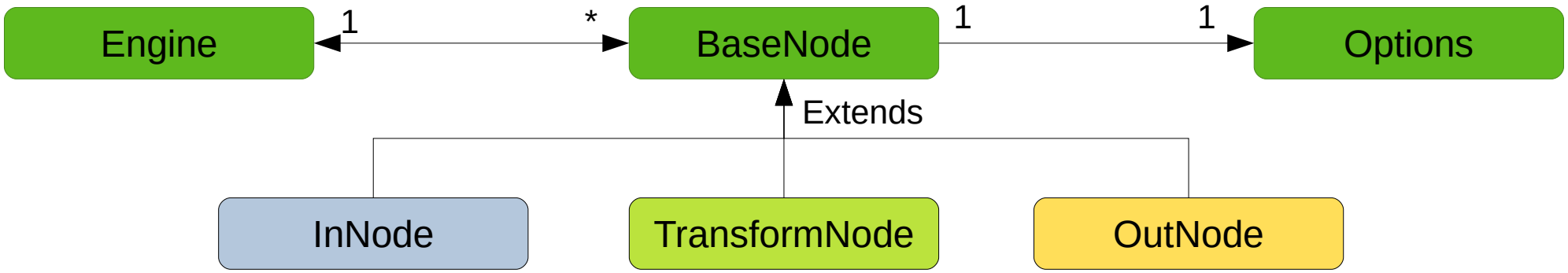
Eventerpretor: Basic structure



Engine

- Creates and manages all nodes
  - On start the nodes are started in this sequence: OutNodes, TransformNodes, InNodes
  - On stop / shutdown the sequence is the other way round
  - Nodes can be accessed with their names (ConcurrentHashMap)
  - Nodes can access the Engine instance

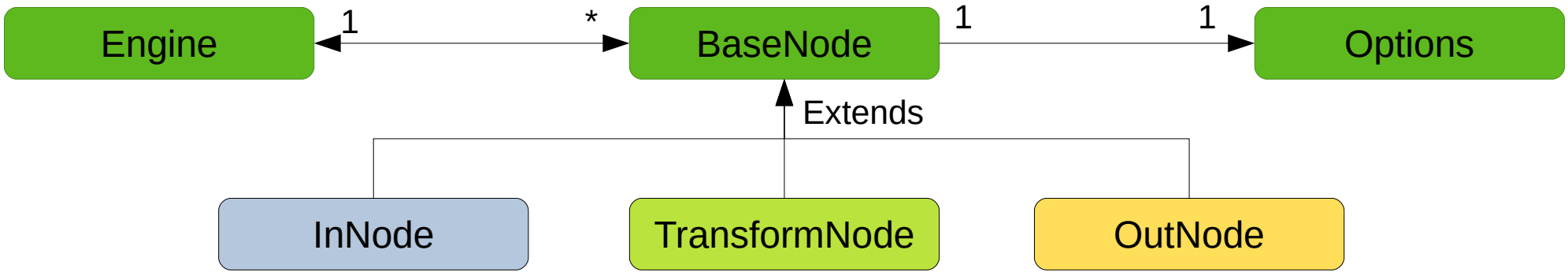
Eventerpretor: Basic structure



Engine

- Provides the main web interface
  - Shows all current nodes and connections
  - Allows restarting of specific nodes to reload their settings
  - Allows clean shutdown

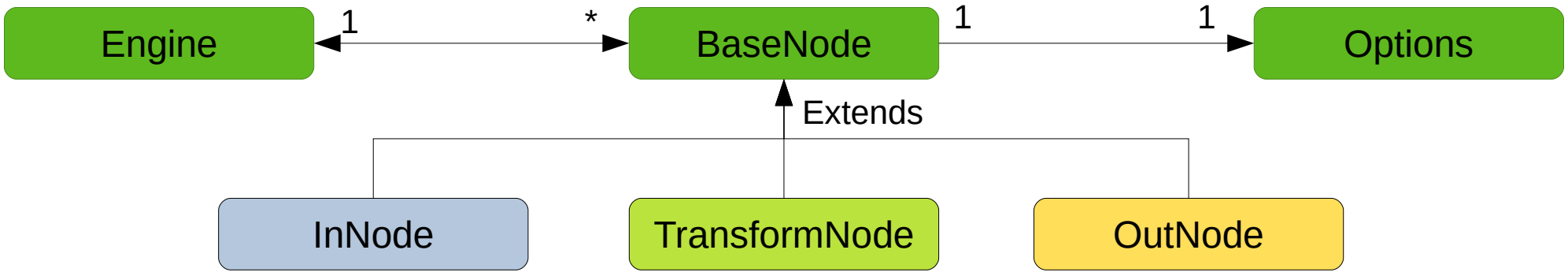
Eventerpretor: Basic structure



BaseNode

- Is the parent class of InNode, TransformNode, OutNode
- One instance of Options is created
- Contains a Thread or Timer depending on what time interval is chosen (0 = Thread)
  - A Thread is intended to be used with blocking functions
  - A Timer should be used especially with InNodes that don‘t receive external data
- Contains abstract functions that have to be implemented by instances of a node type

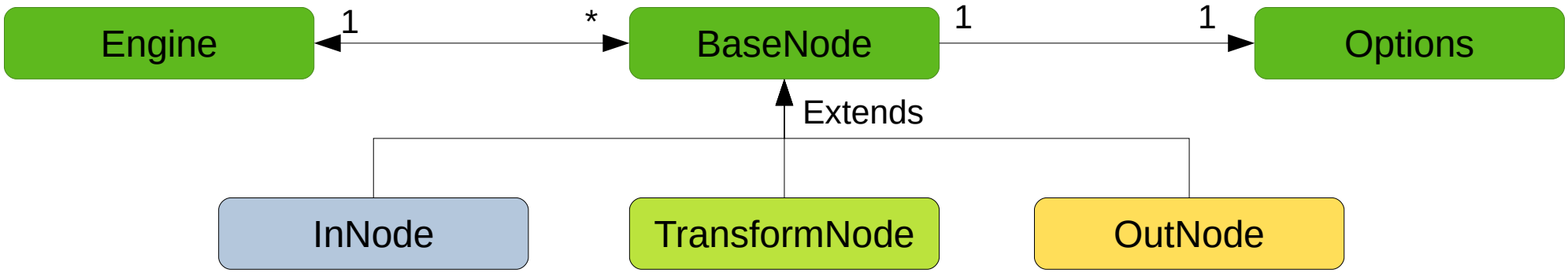
Eventerpretor: Basic structure



InNode / TransformNode / OutNode

- InNode / TransformNode
  - Contain a list of *subscribers* (ConcurrentHashMap)
  - Pass data to *subscribers* by adding it to the LinkedBlockingQueue of each subscriber
- InNodes can‘t receive data from other nodes (no LinkedBlockingQueue)
- OutNodes can‘t send data to other nodes (no subscribers)

Eventerpretor: Basic structure



Options

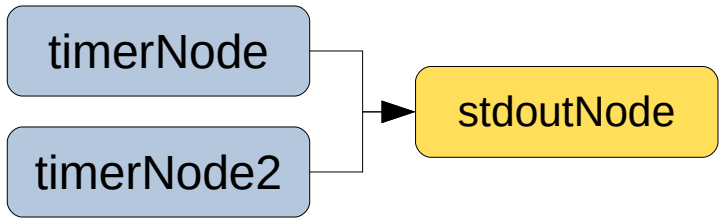
- Each node contains its own *Options* instance and can be accessed by them
- Options are read and stored as json files
- If an options file can‘t be found it is generated with default values provided by a node
- Options can be of different data types
- Options provide a help string

Basic pipeline

Eventerpretor: Basic Pipeline

- Basic pipeline example:

  Print counter of two timers (InTimer)
  to console (OutSTDOUT)



- Required file:
  - pipeline.groovy

- OutSTDOUT doesn‘t provide options
- InTimer option files are automatically generated with default settings at first startup
- The user specified name of the node is used as part of the file name
- The option files can be adapted afterwards
- It is possible to start Eventerpretor in a mode that it just generates option files

Eventerpretor: Basic Pipeline

- Definition: e.g. pipeline.groovy (imports are missing here!)

```
static Boolean init(Engine engine, String[] args) {
    def nodeTimerIn = new InTimer(engine,"timerNode")
    engine.addNode(nodeTimerIn)

    def nodeTimerIn2 = new InTimer(engine,"timerNode2")
    engine.addNode(nodeTimerIn2)

    def nodeStdoutOut = new OutSTDOUT(engine, "stdoutNode")
    engine.addNode(nodeStdoutOut)

    nodeTimerIn.publishTo(nodeStdoutOut)
    nodeTimerIn2.publishTo(nodeStdoutOut)

    return true
}
```

Create node instances (2x InTimer, 1x OutSTDOUT)

Connect both InTimers with OutSTDOUT

Eventerpretor: Basic Pipeline

- Options: options_timerNode.json

```
{
  "interval": {
    "value": 1000,
    "help": "interval in milliseconds",
    "class": "class java.lang.Integer"
  }
}
```

- Options: options_timerNode2.json

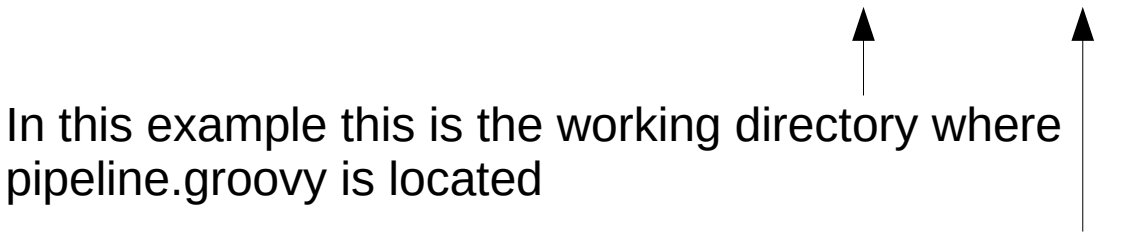
```
{
  "interval": {
    "value": 500,
    "help": "interval in milliseconds",
    "class": "class java.lang.Integer"
  }
}
```

Eventerpretor: Basic Pipeline

Start pipeline with Eventerpretor

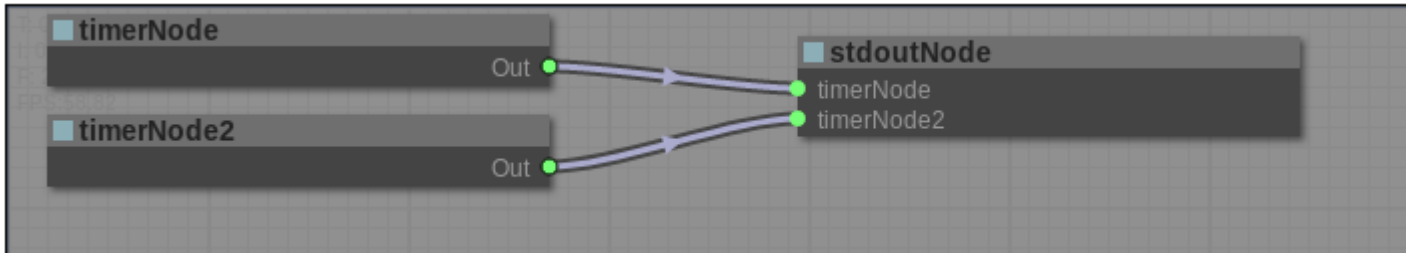
- Assuming the Groovy pipeline script is located in the current directory and the
- Eventerpretor binaries are in: Eventerpretor-Alpha-0.1/ (relative to the current path)

- Eventerpretor-Alpha-0.1/bin/Eventerpretor pipeline.groovy /home/pi/pipelines 0



Eventerpretor: Basic Pipeline

http://localhost:8080



read only view of pipeline(s)

Stop | Restart node

restart a node with the entered name to apply new options

shutdown Eventerpretor