Into the ROS

Advanced ROS Network Introspection

Praxis der Softwarentwicklung
Summerterm 2014

S o f t w a r e d e s i g n

Client

KIT - Karlsruher Institut für Technologie
Fakultät für Informatik
Institut für Anthropromatik und Robotik (IAR)
Intelligente Prozessautomation und Robotik (IPR)

Advisor: Andreas Bihlmaier
andreas.bihlmaier@gmx.net

Contributors

Name	E-Mail-address
Alex Weber	alex.weber3@gmx.net
Matthias Hadlich	matthias.hadlich@student.kit.edu
Matthias Klatte	matthias.klatte@go4more.de
Micha Wetzel	micha.wetzel@student.kit.edu
Sebastian Kneipp	sebastian.kneipp@gmx.net

Karlsruhe, 26.06.2014

Contents

1 Composition

1.1 Architecture

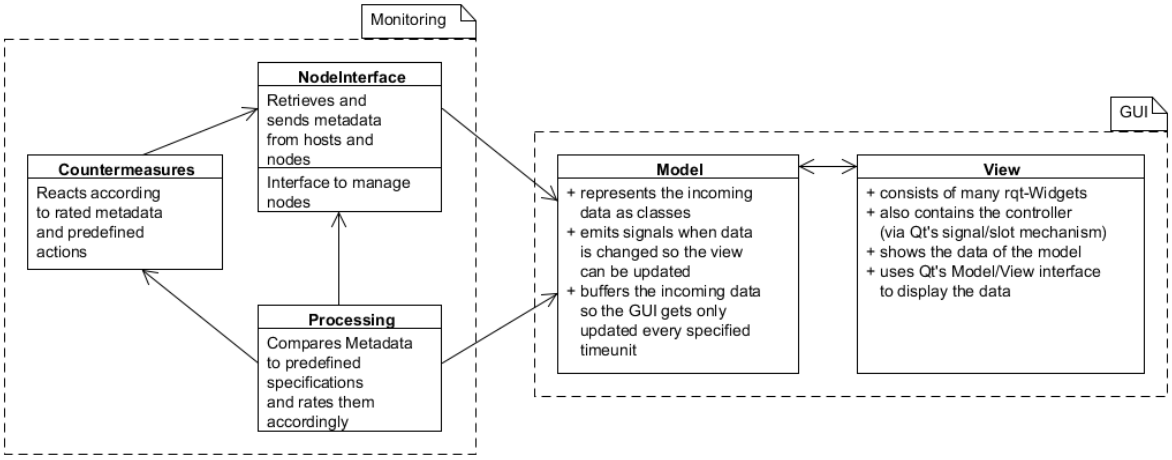


Figure 1.1: architecture

Figure 1.1 shows the general architecture of our software. It is divided into two parts, one for the graphical user interface and one for the monitoring aspect. The right part depicts the GUI. It is designed using the Qt MVC architecture, consisting of only two elements because Qt takes care of the controller: model and view. It will handle user-interaction. The left part depicts the monitoring aspect. It consists of three elements : NodeInterface, Countermeasure and Processing. It will take care of collecting metadata, processing it and taking appropriate action in case of an error.

1.1.1 Monitoring

NodeInterface

- Retrieves and sends metadata from hosts and nodes
- Interface to manage nodes

Processing Compares Metadata to predefined specifications and rates the accordingly

Countermeasures Reacts according to rated metadata and predefined actions

1.1.2 GUI

Model

- Represents the incoming data as classes
- Emits signals when data is changed so the view can be updated
- Buffers the incoming data so the GUI gets only updated every specified timeunit

View

- Consists of many rqt-Widgets
- Also contains the controller (via Qt's signal/slot mechanism)
- Shows the data of the model
- Uses Qt's Model/View interface to display the data
- Also uses pyqtgraph to dynamically plot the incoming data

2 Classes Description

2.1 Processing

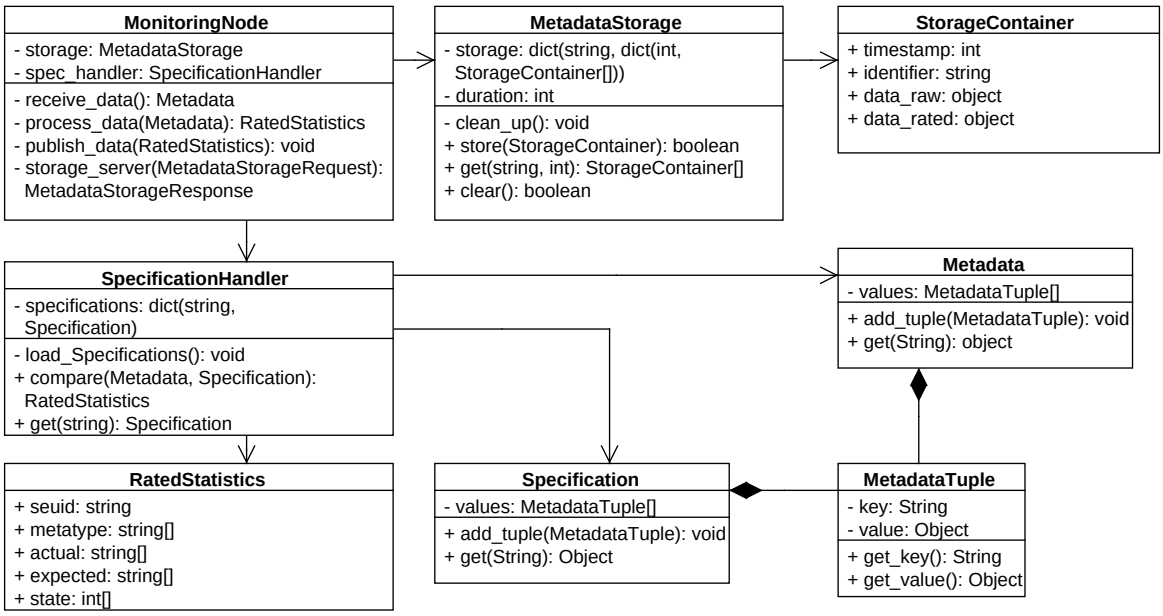
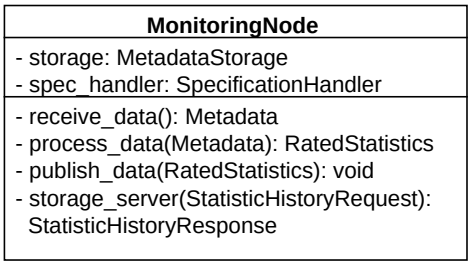


Figure 2.1: The UML diagram of the processing package

2.1.1 MonitoringNode



Main Class wrapping the processing functionality.

Figure 2.2: The MonitoringNode

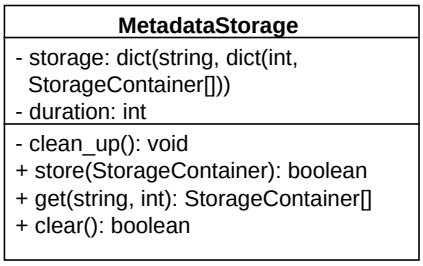
Attributes

- **private MetadataStorage storage**
- **private SpecificationHandler specHandler**

Methods

- **private Metadata receive_data()**
  Receives data incoming from the Subscriber and converts them to Metadata objects.
- **private RatedStatistics process_data(Metadata)**
  Returns the specHandler's compare result
- **private void publish_data(RatedStatistics)**
  Publishes results of the comparison as rated Metadata
- **private StatisticHistoryResponse storage_server(StatisticHistoryRequest)**
  Listens for the GUI Model service calls and returns requested metadata from the storage

2.1.2 MetadataStorage



Saves received metadata packages for a given period of time and can provide them on request.

Figure 2.3: The MetadataStorage

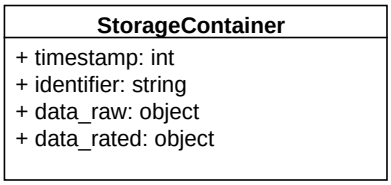
Attributes

- **private dict(string, dict(int, StorageContainer[])) storage**
  Datastructure to store Packages by key and timestamp.
- **private int duration**
  The duration in seconds for data to be stored.

Methods

- **private void clean_up()**
  Deletes Metadata exceeding the duration to store
- **public boolean store(StorageContainer)**
  Stores a given Metadata
- **public StorageContainer[] get(string, int)**
  Returns all Metadata packages for the given connection/host of the given amount of time.
- **public boolean clear()**
  Clears the whole storage

2.1.3 StorageContainer



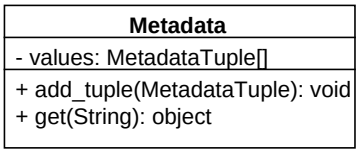
Wraps Metadata in raw and rated form with an identifier and a timestamp. Object to be returned on request by the GUI model.

Figure 2.4: The StorageContainer

Attributes

- **public int timestamp**
  Time when the data came from the subscriber.
- **public string identifier**
  Host/Node/Connection identifier
- **public object data_raw**
  The data as it reaches the subscriber from nodes and hosts.
- **public object data_rated**
  The data like it would be published after being rated.

2.1.4 Metadata



Wraps metadata of exactly one host or node, a topic or a node-topic-combination.

Figure 2.5: The Metadata

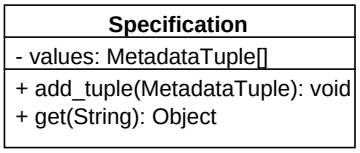
Attributes

- **private MetadataTuple[] values**
  Collection of Metadata regarding multiple meassurements.

Methods

- **public void add_tuple(MetadataTuple)**
  Add a MetadataTuple of information to the bundle.
- **public object get(String)**
  Returns the value of the MetadataTuple with the given key. False, if the key does not exist.

2.1.5 Specification



An object loaded from the specification configurations and basis for comparison of Metadata with desired values.

Figure 2.6: The Specification

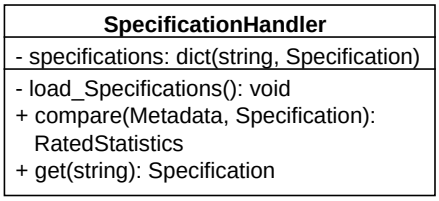
Attributes

- **private MetadataTuple[] values**
  Collection of MetadataTuple objects providing limits for multiple fields.

Methods

- **public void add_tuple(MetadataTuple)**
  Adds a MetadataTuple to the bundle
- **public Object get(String)**
  Returns the value of the MetadataTuple with the given key. The returned value would be a list containing limit values for the most measured fields. False, if the key does not exist.

2.1.6 SpecificationHandler



Loads the specifications from the parameter server and compares them to the actual metadata.

Figure 2.7: The SpecificationHandler

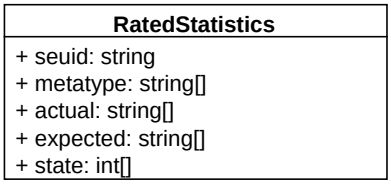
Attributes

- **private dict(string, Specification) specifications**
  Datastructure to keep all loaded Specification objects

Methods

- **private void load_specifications()**
  Loads the specifications from configuration files into Specification objects and stores them
- **public RatedStatistics compare(Metadata, Specification)**
  Compares a given Metadata object with a given Specification object regarding all available fields. Returns a RatedStatistics object wrapping potential divergences.
- **public Specification get(string)**
  Returns the specification for a given identifier

2.1.7 RatedStatistics



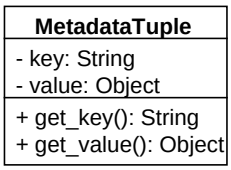
Wraps the result of the comparison between the actual metadata and the specificaton.

Figure 2.8: The RatedStatistics

Attributes

- **public string seuid**
  Identifies the node/host/connection
- **public string[] metatype**
  The metadata that was out of bounds
- **public string[] actual**
  The actual values
- **public string[] expected**
  The expected values
- **public int[] state**
  State of the metadata from the node/host/connection : state: 0 = high ; 1 = low ; 2 = unknown

2.1.8 MetadataTuple



Stores any kind of value for a certain key. Specifications storing values indicating limits, Metadata storing absolute actual values.

Figure 2.9: The MetadataTuple

Attributes

- **private String key**
- **private Object value**

Methods

- **public String get_key()**
- **public Object get_value()**

2.2 NodesInterface

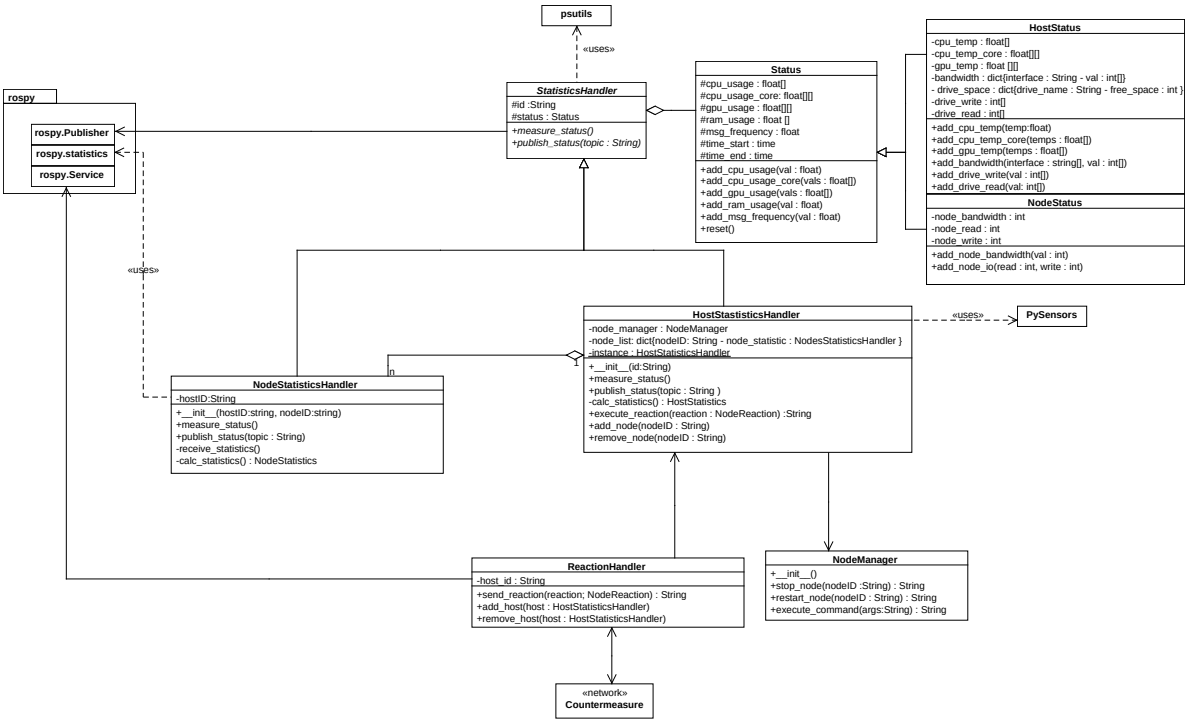
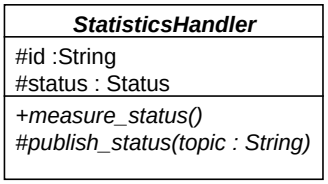


Figure 2.10: UML diagram of the NodeInterface package

2.2.1 StatisticsHandler



Abstract Class to Handle Statistics of Hosts or Nodes.

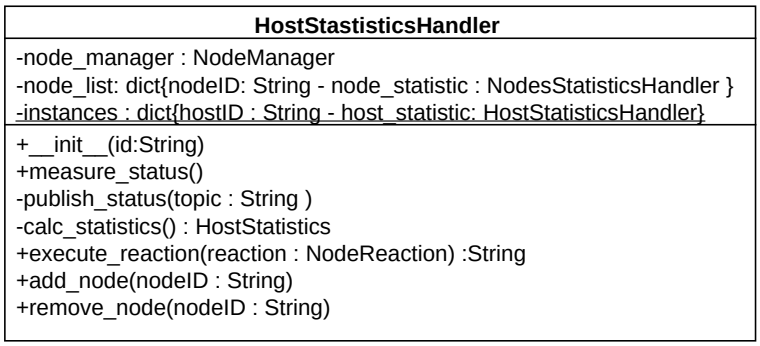
Attributes

- **protected String id**
  Id of the Host or Node.
- **protected Status status**
  Holds the current status.

Methods

- **public void measure_status()**
  Collects information about the current status using psutils. Triggered periodically.
- **public void publish_status(String topic)**
  Publishes the current status to a topic using ROS's publisher-subscriber mechanism.

2.2.2 HostStatisticsHandler



Represents a host . Limited to one instance per host. Collects statistics about the current state of the host and sends them using the publisher-subscriber mechanism.

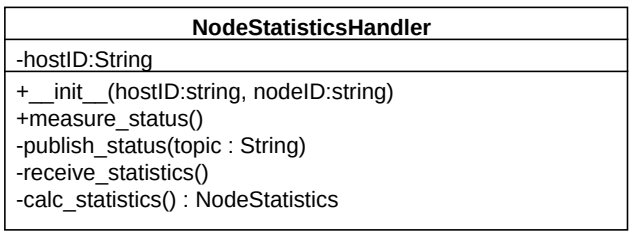
Attributes

- **private NodeManager node_manager**
  NodeManager providing function to restart and stop Nodes.
- **private dict{String nodeID - NodesStatisticsHandler node_statistic } node_list**
  Dictionary holding all Nodes and their statistics, currently running on the host.
- **private static dict{String hostID - HostStatisticsHandler host_statistic }**
  Holds references to all initiated host, to prevent multiple instances of a single host.

Methods

- **public void measure_status()**
  Collects information about the host's current status using psutils. Triggered periodically.
- **public void publish_status(String topic)**
  Publishes the current status to a topic using ROS's publisher-subscriber mechanism. Triggered periodically.
- **public String execute_reaction(NodeReaction reaction)**
  Parses through the reaction and calls the appropriate method from the NodeManager. Returns a message about operation's success.
- **public void add_node(String nodeID)**
  Adds a Node with the given id to the host.
- **public void remove_node(String nodeID)**
  Removes the Node with the given id from the host.
- **private HostStatistics calc_statistics**
  Calculates statistics like mean, standard deviation and max from the status. Returns an instance of HostStatistics which can be published.

2.2.3 NodeStatisticsHandler



Holds the statistics of an individual Node.

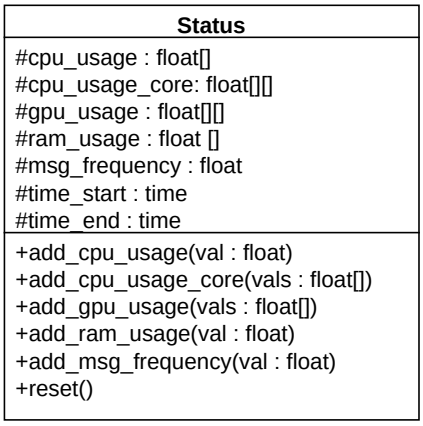
Attributes

- **private String hostID**
  Id of the host this node runs on.

Methods

- **public void measure_status()**
  Collects information about the node's current status using psutils and rospy.statistics Triggered periodically.
- **public void publish_status()**
  Publishes the current status to a topic using ROS's publisher-subscriber mechanism. Triggered periodically.
- **private void receive_statistics()**
  Receives the statistics published by ROS Topic statistics
- **private NodeStatistic calc_statistics**
  Calculates statistics like mean, standard deviation and max from the status. Returns an instance of NodeStatistic which can be published.

2.2.4 Status



Container Class to Store information about the current status.

Attributes

- **protected float[] cpu_usage**
  Percentage of the cpu used.

- **protected float[][] cpu_usage_core**
  Percentage of the cpu used per core.
- **protected float[] ram_usage**
  Percentage of ram used.
- **protected int msg_frequency**
  Frequency of network calls.
- **protected time time_start**
  Time of the start of the measurements.
- **protected time time_end**
  Time of the end of the measurements.

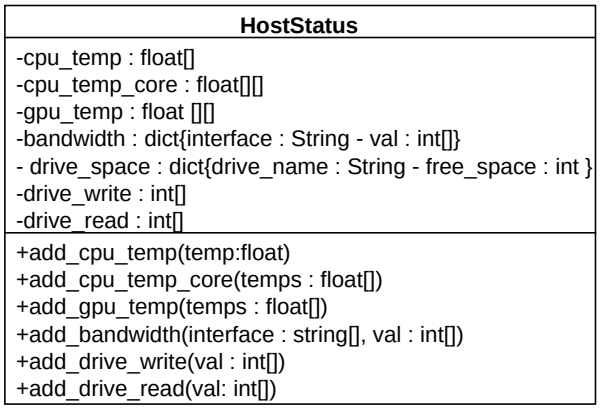
Methods

- **public void add_cpu_usage(float value**
  Adds another measured value to cpu_usage.
- **public void add_cpu_usage_core(float[] values)**
  Adds another measured value to per core cpu_usage_core.
- **public void add_gpu_usage(float[] values**
  Adds another measured value per card to gpu_usage.
- **public void add_ram_usage(float value)**
  Adds another measured value to ram_usage.
- **public void add_msg_frequency(float value)**
  Adds another measured value to msg_frequency.
- **public void reset()**
  Resets the status.

2.2.5 HostStatus

Attributes

- **private float[] cpu_temp**
  Current CPU temperature in Celsius.
- **private float[][] cpu_temp_core**
  Current CPU temperature per core in Celsius.



Extension of Status , to store additional information used by hosts.

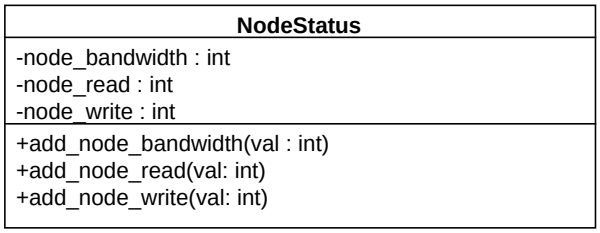
- **private float[][] gpu_temp_core**
  Current GPU temperature per GPU in Celsius.
- **private dict{String interface - int[] bytes} bandwidth**
  Bytes sent through a NIC since the start of the time window.
- **private dict{String drive - int[] free_space} bandwidth**
  Free Space per drive.
- **private int[] drive_write**
  Bytes written per drive since the start of the time window.
- **private int[] drive_read**
  Bytes read per drive since the start of the time window.

Methods

- **public void add_cpu_temp(float temp)**
  Adds another measured value to cpu_temp.
- **public void add_cpu_temp_core(float[] temps)**
  Adds another measured value per core to cpu_temp_core.
- **public void add_gpu_temp(float[] temps)**
  Adds another measured value per card to gpu_temp.
- **public void add_bandwidth(String[] Interface, int[] bandwidth)**
  Adds another measured value to bandwidth.
- **public void add_drive_write(int[] byte)**
  Adds another measured value per drive to drive_write.

- **public void add_drive_read(int[] byte)**
  Adds another measured value per drive to drive_read.
- **reset()**
  Resets the status.

2.2.6 NodeStatus



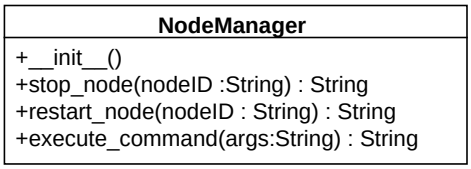
Extension of Status , to store additional information used by nodes.

Attributes

- **private int node_bandwidth**
  Bytes the node has sent through the network.
- **private int node_read**
  Bytes the node has read from a hard drive since the start of the time window.
- **private int node_write**
  Bytes the node has written to a hard drive since the start of the time window.

Methods

- **public void add_node_bandwidth(int byte)**
  Adds another measured value to node_bandwidth.
- **public void add_node_io(int read, int write)**
  Adds another pair of measured read and write values to node_read and node_write.
- **reset()**
  Resets the status.



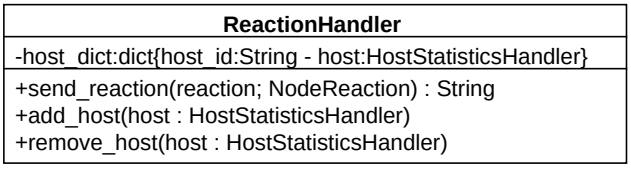
Can restart or stop nodes or execute a countermeasure.

2.2.7 NodeManager

Methods

- **public String stop_node(String nodeID)**
  Stops the node with the given id. Returns a message about operation's success.
- **public String restart_node(String nodeID)**
  Restarts a node with the given id. Returns a message about operation's success.
- **public String execute_command(String[] args)**
  Executes a system call with the given arguments. Returns a message about operation's success.

2.2.8 ReactionHandler



Delegates the countermeasure to the concerned host.

Attributes

- **private dict{String host_id - HostStatisticsHandler host } host_dict**
  Dictionary of all hosts running on the network

Methods

- **public String send_reaction(NodeReaction reaction)**
  Parses the reaction and delegates it to the concerned host. Returns a message about operation's success using rospy.Service

- **public void add_host(HostStatisticsHandler host)**
  Adds a host to the dictionary
- **public void remove_host(HostStatisticsHandler host)**
  Removes a host from the dictionary

2.2.9 psutils

Library to acquire the system's usage statistics. For a more in-depth documentation, see the official psutils documentation.

Used methods

- **psutil.cpu_percent(interval, boolean percpu)**
  Return a float representing the current system-wide CPU usage.
- **psutil.virtual_memory()**
  Return statistics about system memory usage.
- **psutil.net_io_counters(boolean pernic)**
  Return system-wide network I/O statistics.
- **psutil.disk_usage(String path)**
  Return disk usage statistics about the given path.
- **psutil.disk_io_counters(boolean perdisk)** Return system-wide disk I/O statistics.
- **psutil.disk_partitions()**
  Return all mounted disk partitions as a list of namedtuples.
- **psutil.Process(pid)**
  Represents an process with the given pid

2.3 Countermeasure

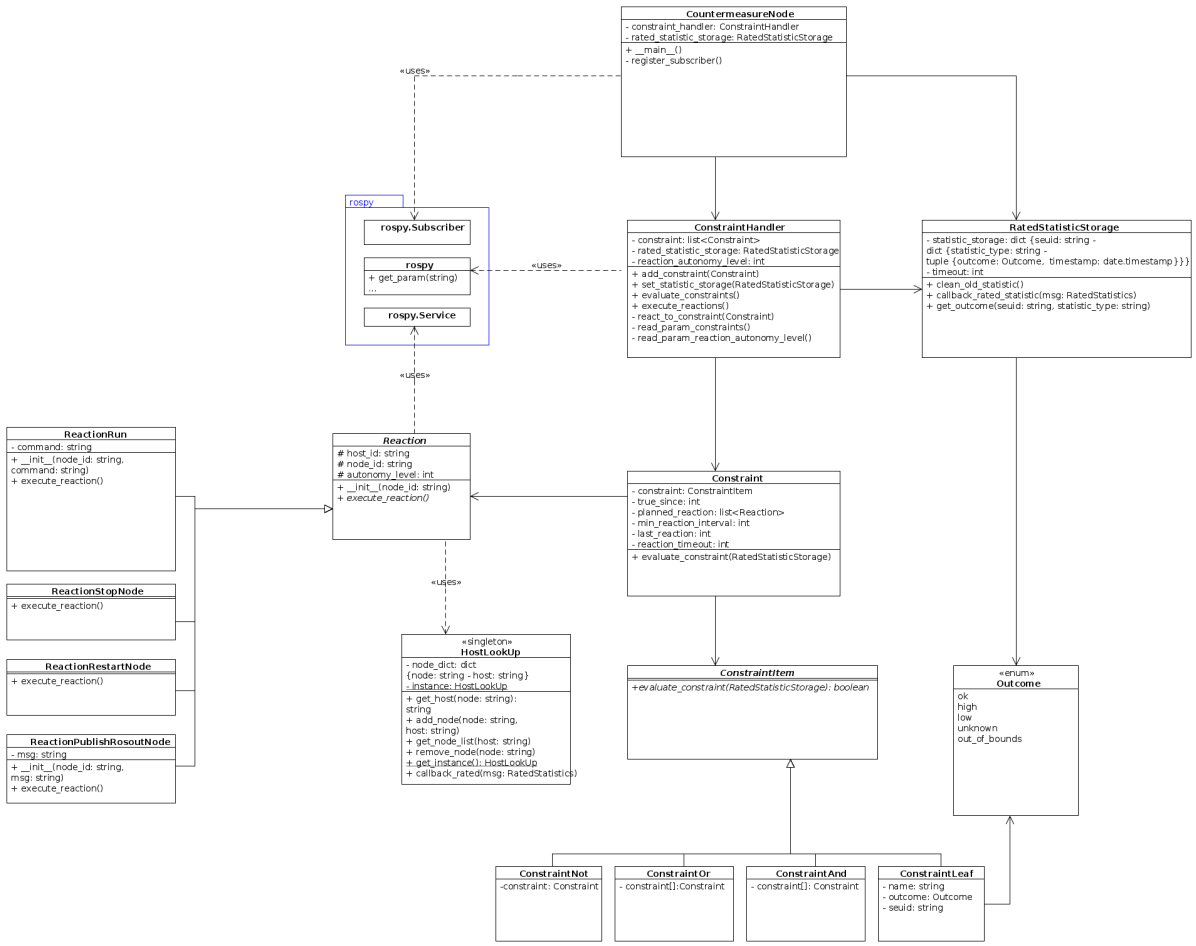
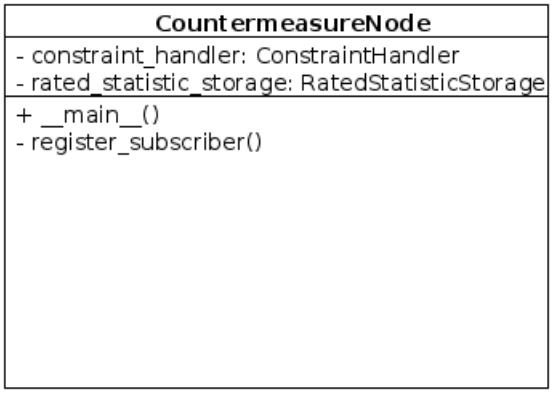


Figure 2.11: The UML diagram of the countermeasure package

2.3.1 CountermeasureNode



A ROS node. Evaluates incoming rated statistics with a list of constraints. If those constraints turn out to be true appropriate action is taken.

Figure 2.12: The CountermeasureNode

Attributes

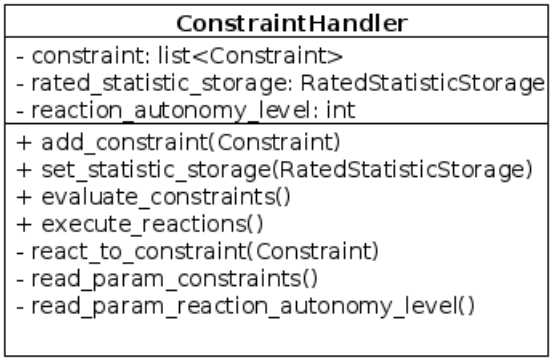
- **private ConstraintHandler constraint_handler**
  The handler for all constraints.
- **private RatedStatisticStorage rated_statistic_storage**
  The storage of all incoming rated statistic.

Methods

- **public void __main__()**

  Periodically (threading) evaluates the constraints and cleans old statistics.
- **private void register_subscriber()**
  Registers to the rated statistics.

2.3.2 ConstraintHandler



Manages all constraints, checks if they are true and executes appropriate reactions if neccessary.

Figure 2.13: The ConstraintHandler

Attributes

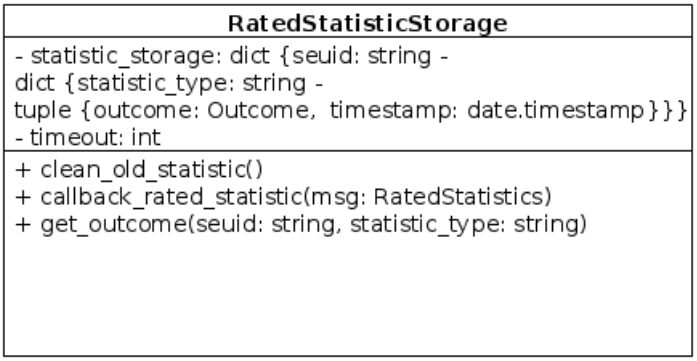
- **private list<Constraint> constraint_list**
  Contains a list of all constraints.
- **private RatedStatisticStorage rated_statistic_storage**
  Contains all incoming rated statistic.
- **private int reaction_autonomy_level**
  Only reactions with an autonomy_level <= reaction_autonomy_level get executed.

Methods

- **public void add_constraint(Constraint)**
  Adds an constraint to this list.
- **public void set_statistic(RatedStatisticStorage)**
  Sets the Statistic to use. Should only be needed on initialisation.
- **public void evaluate_constraints()**
  Evaluates every constraint.
- **public void execute_reactions()**
  Checks if there are any new reactions to do and executes them.
- **private void react_to_constraint(Constraint)**
  Executes an single Reaction and updates the attributes of the Constraint.

- **private void read_param_constraints()**
  Reads all constraints from the parameter server.
- **private void read_param_reaction_autonomy_level()**
  Reads the reaction_autonomy_level from the parameter server.

2.3.3 RatedStatisticStorage



A database which contains the current state of all rated statistics.

Figure 2.14: The RatedStatisticStorage

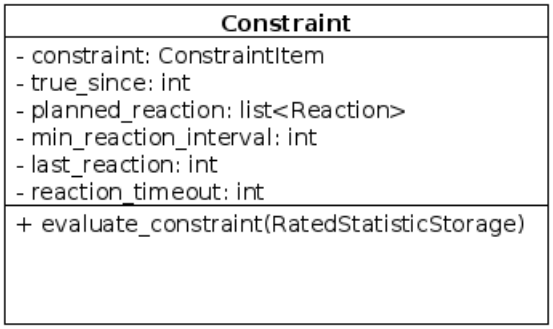
Attributes

- **private dict{string seuid - dict{string statistic_type - tuple{Outcome outcome,date.timestamp timestamp}}} statistic_dict**
  A dictionary containing all rated statistic information with their outcome and an timestamp when they got added / updated to the dictionary.
- **private int timeout**
  The timeout after which an item in ratedstatistic is declared too old and should be removed from the dict.

Methods

- **public void clean_old_statistic()**
  Checks the complete dictionary for statistics older than timeout seconds and removes them.
- **public void callback_rated_statistic(RatedStatistics msg)**
  Callback for incoming rated statistics. Adds them to the dictionary or removes items from the dictionary if the rated statistic says that its within bounds again.
- **public Outcome get_outcome(string seuid, string statistic_type)**
  Returns the outcome of the specific seuid and statistic_type.

2.3.4 Constraint



Contains the whole constraint with corresponding reactions.

Figure 2.15: The Constraint class

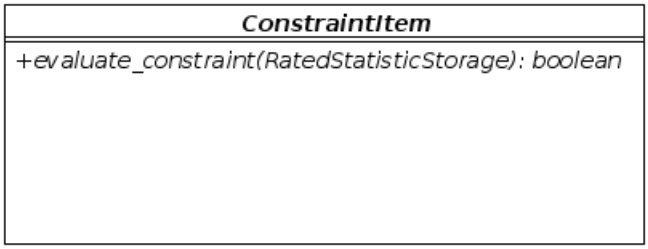
Attributes

- **private ConstraintItem constraint**
  First constraint in the chain of ConstraintItems.
- **private int true_since**
  Epoch time in milliseconds since the constraint is true, if the constraint is not true it is 0.
- **private list<Reaction> planned_reaction**
  An list of reactions that should be executed if the constraint has been true longer than min_reaction_interval milliseconds.
- **private int min_reaction_interval**
  The minimum time needed in ms that the constraint needs to be true to execute the planned_reaction.
- **private int last_reaction**
  Contains the epoch time in ms when the reaction corresponding to this constraint has been executed for the last time. It is 0 if it has never been executed.
- **private int reaction_timeout**
  Minimum durotation in ms needed before an reaction can happen again.

Methods

- **public void evaluate_constraint(RatedStatisticStorage)**
  Evaluates this constraint and sets the attributes according to the result of the evaluation.

2.3.5 ConstraintItem



Abstract description of a Constraint, can be a logical operation on constraints or an actual constraint.

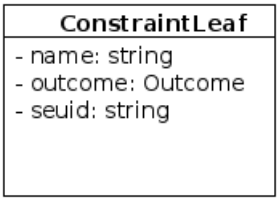
Figure 2.16: The ConstraintItem class

Attributes

Methods

- **public abstract boolean evaluate_constraint(RatedStatisticStorage)**
  Evaluates if this constraint, given the available RatedStatisticStorage, is true.

2.3.6 ConstraintLeaf



Contains an actual statistic datapoint and the seuid the datapoint belongs to.

Figure 2.17: The ConstraintLeaf class

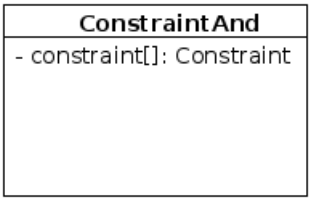
Attributes

- **private string name**
  Contains the name of the statistic data.
- **private Outcome outcome**
  Contains the outcome needed for this constraint to be true.
- **private string seuid**
  Contains the unique identifier of the corresponding StatisticEntity.

Methods

- **public abstract boolean evaluate_constraint(RatedStatisticStorage)**
  Returns true if this constrain is true for the RatedStatisticStorage.

2.3.7 ConstraintAnd



An constraints consisting of other constraints logically and colligated.

Figure 2.18: The ConstraintAnd class

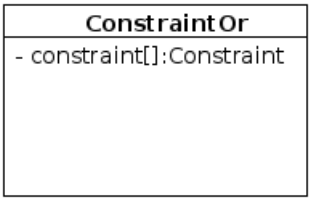
Attributes

- **private Constraint[] constraint**
  Contains constraints to be evaluated with an logical and.

Methods

- **public boolean evaluate_constraint(RatedStatisticStorage)**
  Returns true if the evaluation of all constains in the array returns true.

2.3.8 ConstraintOr



An constraints consisting of other constraints logically or colligated.

Figure 2.19: The ConstraintOr class

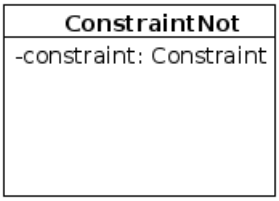
Attributes

- **private Constraint[] constraint**
  Contains constraints to be evaluated with an logical or.

Methods

- **public boolean evaluate_constraint(RatedStatisticStorage)**
  Returns true if the evaluation of at least one constraint returns true.

2.3.9 ConstraintNot



An constraints consisting of another constraint negated.

Figure 2.20: The ConstraintNot class

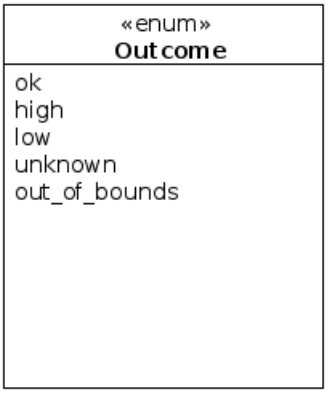
Attributes

- **private Constraint constraint**
  The constraint to be evaluated negated.

Methods

- **public boolean evaluate_constraint(RatedStatisticStorage)**
  Returns true if the evaluation of the constraint returns false.

2.3.10 Enum Outcome



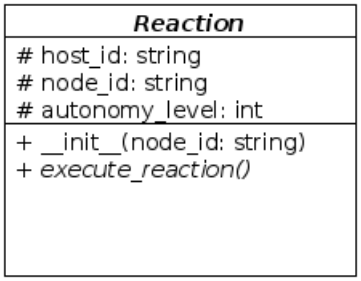
An enumeration of all states an rated statistic can have.

Figure 2.21: The Outcome enum

Types

- **high**
  Data value is too high.
- **low**
  Data value is too low.
- **unknown**
  Data value is unknown.
- **out_of_bounds**
  Data value is either too high or too low.

2.3.11 Reaction



Abstract Reaction to an Constraint. The Reaction is to be executed on the corresponding host of the given node.

Figure 2.22: The Reaction class

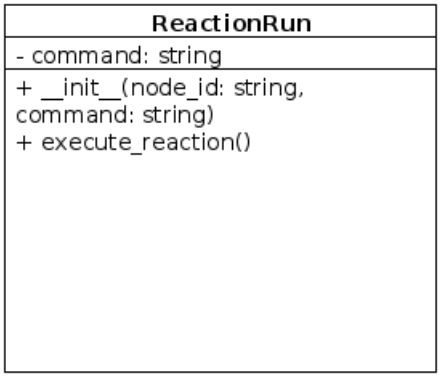
Attributes

- **protected string host_id**
  Contains the host on which the node is run on.
- **protected string node_id**
  The id of the node the reaction is ment to act upon.
- **protected int autonomy_level**
  This constraint only gets evaluatet if the autonomy_level is <= reaction_autonomy_level.

Methods

- **public void __init__(string node_id)**
  Initializes the reaction. Sets the node to execute the reaction on. finds the corresponding host to the given node.
- **public void execute_reaction()**
  Executes the reaction as a service call to the HostStatistic node.

2.3.12 ReactionRun



An Reaction which executes a command on the remote machine the specified node runs on.

Figure 2.23: The ReactionRun class

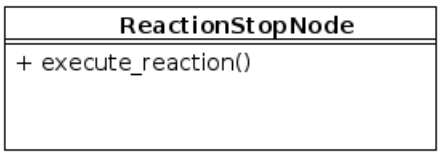
Attributes

- **private string command**
  Contains the command to be executed.

Methods

- **public void __init__(string node_id,string command)**
  Initializes the reaction. Set the command to be executed.
- **public void executeReaction()**

2.3.13 ReactionStopNode



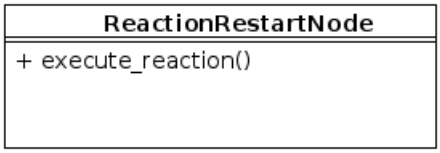
A reaction that is able to stop an node.

Figure 2.24: The ReactionStopNode class.

Methods

- **public void exececute_reaction()**

2.3.14 ReactionRestartNode



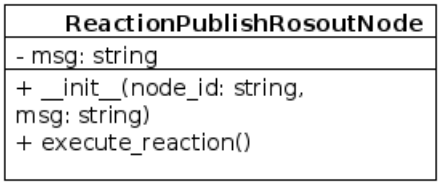
A reaction that is able to restart an node.

Figure 2.25: The ReactionRestartNode class.

Methods

- **public void exececute_reaction()**

2.3.15 ReactionPublishRosoutNode



A reaction that is able to publish a message on rosout.

Figure 2.26: The ReactionPublishRosoutNode class.

Attributes

- **private string msg**
  The message to be published.

Methods

- **public void __init__(string node_id,string msg)**
  Initializes the reaction. Set the message to be published on rosout.
- **public void exececute_reaction()**

2.3.16 HostLookUp

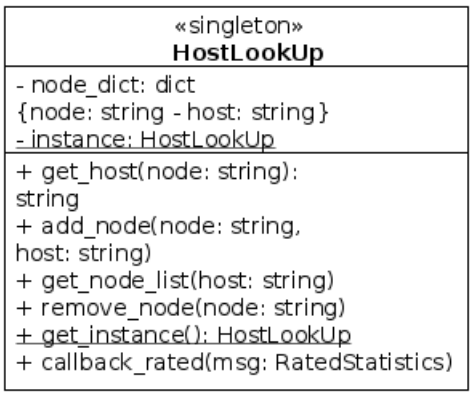


Figure 2.27: The HostLookUp class

Singleton. Contains a dictionary of all nodes and the hosts they run on. Works only for nodes which are on an host who has an HostStatisticNode running.

Attributes

- **private dict{string node - string host} node_dict**
  Contains all nodes which are on an host who has an HostStatisticNode running. Host is the host the node runs on.
- **private static HostLookUp instance**
  The singleton instance.

Methods

- **public string get_host(string node)**
  Returns the host the node runs on.
- **public void add_node(string node, string host)**
  Adds an node - host tuple to the dictionary.
- **public list<string> get_node_list(string host_id)**
  Returns all nodes of a specifid host.
- **public void remove_node(string node)**
  Removes an node from the dictionary.
- **public static HostLookUp get_instance()**
  Returns the instance of HostLookUp.

- **public void callback_rated(RatedStatistics msg)**
  Callback for rated statistics. Adds unseen nodes to the dictionary.

2.4 GUI

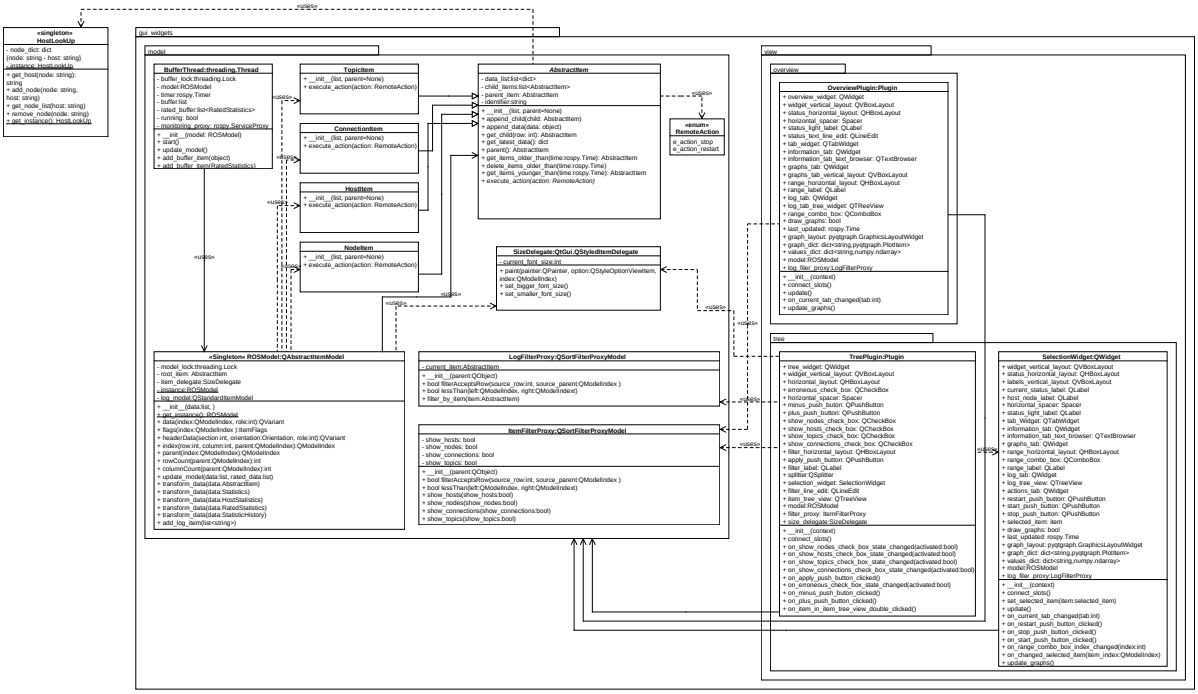


Figure 2.28: The GUI class diagram

2.5 GUI - Model

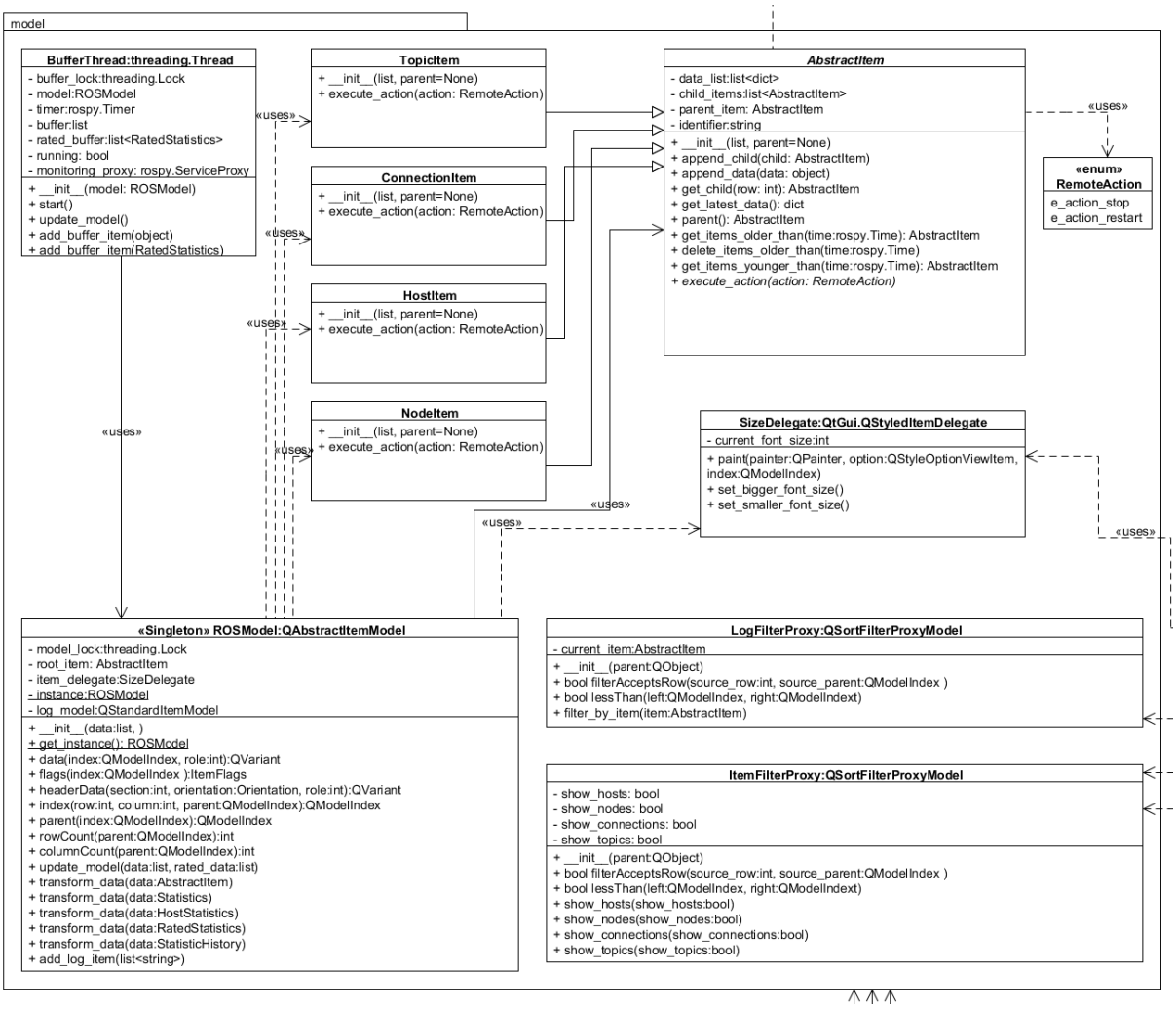
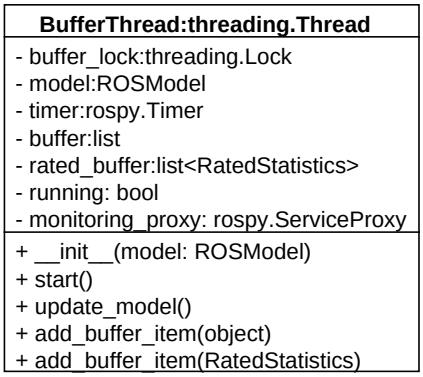


Figure 2.29: The model class diagram

2.5.1 BufferThread



This thread should buffer the incoming data from the topics and regulary update the model.

Figure 2.30: The BufferThread

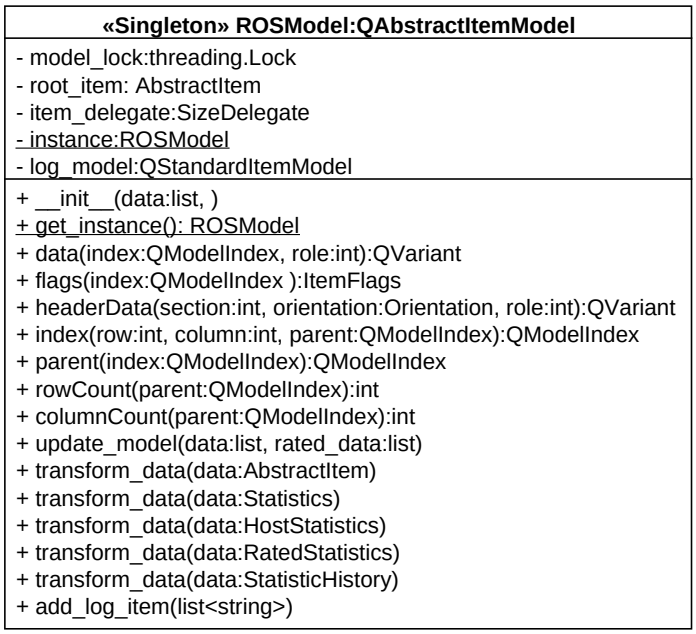
Attributes

- **private threading.Lock buffer_lock**
  The lock that guards the buffer from getting modified parallely
- **private ROSModel model**
  The model of the hosts/nodes/topics/connections
- **private rospy.Timer timer**
  ROS Timer which regularily calls update_model()
- **private list buffer**
  Buffers the tons of incomming data by simply storing it here together with a timestamp for later usage
- **private okcdob_oqq:rospy.OkcdobOqq**
  Drsc sc kx okcdob oqq. Myxqbkdevkdsyxc.
- **private list<RatedStatistics> rated_buffer**
  A list for the incoming RatedStatistics items, stored here for later processing.
- **private bool running**
  Is true if the thread is running
- **private rospy.ServiceProxy monitoring_proxy**
  The proxy to the monitoring node for obtaining statistics and rated statistics of the past minutes. To be called only once when the GUI started and the MonitoringNode has been running for a while

Methods

- **public void start()**
  Starts the thread and also the timer for regulary updates of the model. It is ensured via the running attribute that this function cannot be called multiple times.
- **public void update_model()**
  Starts the update of the model. Will be called regulary by the timer. Will first read the data from the buffer and add the according data items to the items of the model and afterwards use the rated_buffer to add a rating to these entries.
- **public void add_buffer_item(object message)**
  Adds the item to the buffer list. Will be called whenever data from the topics is available.
- **public void add_buffer_item(RatedStatistics message)**
  Adds the RatedStatistics item to the rated_buffer

2.5.2 ROSModel



Represents the data as a QtModel. This enables automated updates of the View.

Figure 2.31: The ROSModel

Attributes

- **private threading.Lock model_lock**
  Protects the model from parallel modification
- **private AbstractItem root_item**
  Contains the list of headers
- **private SizeDelegate item_delegate**
  The item_delegate is responsible for painting the rows in a specific manner e.g. to make the font bigger.
- **private QStandardItemModel log_model**
  The model which is used for representing the log data
- **private static ROSModel instance**
  The static instance for the Singleton

Methods

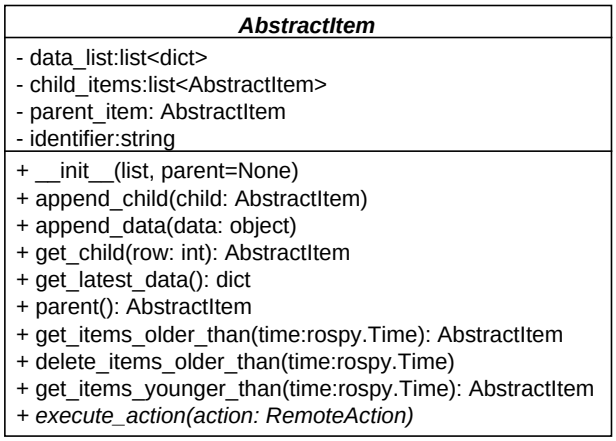
- **public __init__()**
  Defines the class attributes especially the root_item which later contains the list of headers e.g. for a TreeView representation
- **public ROSModel get_instance()**
  Returns the instance of the ROSModel
- **public QVariant data(QModelIndex index, int role)**
  Returns the data of an item at the given index
- **public ItemFlags flags(QModelIndex index)**
  Returns the flags of the item at the given index (like Qt::ItemIsEnabled)
- **public QVariant headerData(int section, Orientation orientation, int role)**
  Returns the headerData at the given section
- **public QModelIndex index(int row, int column, QModelIndex parent)**
  Returns the index of an item at the given column/row
- **public QModelIndex parent(QModelIndex index)**
  Returns the QModelIndex of the parent of the child item specified via its index
- **public int rowCount(QModelIndex index)**
  Returns the amount of rows in the model

- **public int columnCount(QModelIndex index)**
  Returns the amount of columns in the model
- **public void update_model(list data, list ratd_data)**
  Updates the model by using the items of the list. The items will be of the message types
- **public void transform_data(Statistics data)**
  Integrates a TopicStatistics in the model by modifing its item/s by adding a new dict to the corresponding item (especially the TopicItem and the ConnectionItem)
- **public void transform_data(NodeStatistics data)**
  Integrates a NodeStatistics in the model by modifing its item/s by adding a new dict with the entries of the given parameter
- **public void transform_data(HostStatistics data)**
  Integrates a HostStatistics in the model by modifing its item/s by adding a new dict with the entries of the given parameter
- **public void transform_data(RatedStatistics data)**
  Add the rating to an existing entry by modifing the dict of the corresponding item/s
- **public void transform_data(StatisticHistory data)**
  When using the monitor_proxy to receive about the last minutes from the monitoring node it returns a StaticHistory item which can then be integrated in the model via this method
- **public void add_log_item(list<string>)**
  Adds the given list as a log entry to the model

2.5.3 AbstractItem

Attributes

- **private list<dict> data_list**
  Contains the data of the abstract item including a time stamp so that the progress in time can be shown
- **private list<AbstractItem> child_items**
  The childs of this item
- **private AbstractItem parent_item**
  The parent of this item



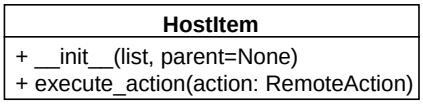
Provides a unified interface to access the items of a model.

Figure 2.32: The AbstractItem

Methods

- **public void append_child(AbstractItem child)**
  Append a child to the list of childs
- **public void append_data(oject data)**
  Append data to the data_list of the AbstractItem
- **public AbstractItem get_child(int row)**
  Returns the child at the position row
- **public dict get_latest_data()**
  Returns the latest dict of the data_list
- **public AbstractItem parent()**
  Returns the parent of this or None if there is none
- **public AbstractItem get_items_older_than(rospy.Time time)**
  Returns all items wich are older than *rospy.Time*
- **public void delete_items_older_than(rospy.Time time**)
  Deletes all items wich are older than *rospy.Time*
- **public AbstractItem get_items_younger_than(rospy.Time)**
  Returns all items wich are younger than *rospy.Time time*
- **public abstract void execute_action(RemoteAction action)**
  Executes a action on the current item like stop or restart. Calls to this method should be redirected to the remote host on executed there.

2.5.4 HostItem



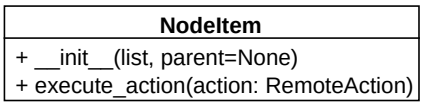
A HostItem represents a host with all its data.

Figure 2.33: The HostItem

Methods

- **public execute_action(RemoteAction action)**
  Sends a signal to stop or restart a node

2.5.5 NodeItem



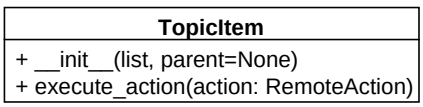
A NodeItem represents a node with all of its data. It also has a interface to start/stop/restart nodes.

Figure 2.34: The NodeItem

Methods

- **public execute_action(RemoteAction action)**
  Sends a signal to stop or restart the node

2.5.6 TopicItem



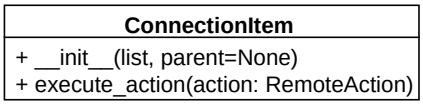
A TopicItem reprensents a specific topic which contains many connections and has attributes like the number of sent messages.

Figure 2.35: The TopicItem

Methods

- **public execute_action(RemoteAction action)**
  Not senseful, throws an exception

2.5.7 ConnectionItem



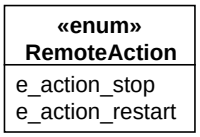
A ConnectionItem reprensents the connection between a publisher and a subscriber and the topic they are puglishing / listenening on.

Figure 2.36: The ConnectionItem

Methods

- **public execute_action(RemoteAction action)**
  Not senseful, throws an exception

2.5.8 Enum RemoteAction



Gives a predefinition for a remote interaction with hosts and nodes.

Figure 2.37: The ROSModel

Types

- **e_action_stop**
  The action that should stop a host or node
- **e_action_restart**
  The action that should restart a host or node

2.5.9 ItemFilterProxy

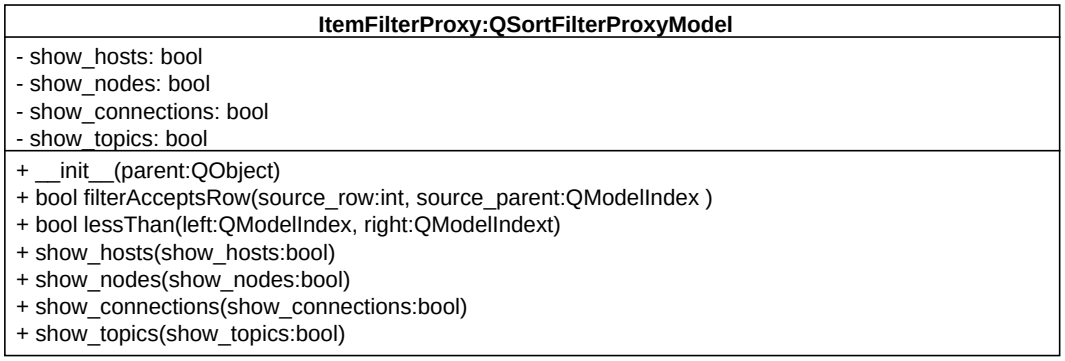


Figure 2.38: The ItemFilterProxy

The ItemFilterProxy which is a QSortFilterProxyModel helps to filter the data going to the view so the user only sees what he wants to see (which he can modified by telling the view).

2.5.10 Attributes

- **private bool show_hosts**
  True if hosts should be shown
- **private bool show_nodes**
  True if nodes should be shown
- **private bool show_connections**
  True if connections should be shown
- **private bool show_topics**
  True if topics should be shown

Methods

- **public bool filterAcceptsRow(int source_row, QModelIndex source_parent)**
  Tells by analysing the given row if it should be shown or not. This behaviour can be modified via the show_* methods or the setFilterRegExp method.
- **public bool lessThan(QModelIndex left, QModelIndex right)**
  Defines the sorting behaviour when comparing two entries of model item by telling how to compare these.

- **public void show_hosts(bool show_hosts)**
  Set true if hosts should be shown
- **public void show_nodes(bool show_nodes)**
  Set true if nodes should be shown
- **public void show_connections(bool show_connections)**
  Set true if connections should be shown
- **public void show_topics(bool show_topics)**
  Set true if topics should be shown

2.5.11 LogFilterProxy

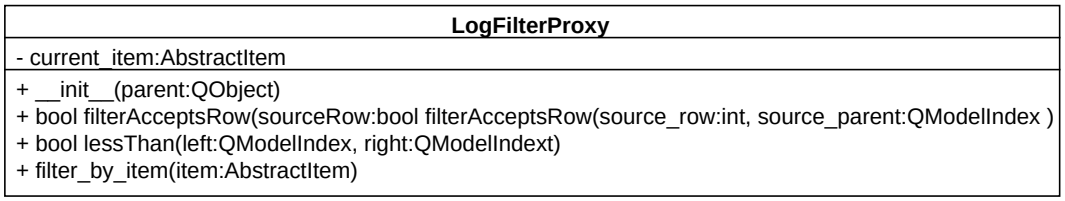


Figure 2.39: The LogFilterProxy

The LogFilterProxy will especially be used to filter the complete log e.g. by a specific node. This function is needed in the SelectionWidget where of course only the log of the current selection should be shown.

2.5.12 Attributes

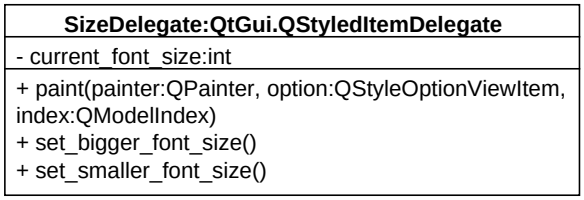
- **private current_item: AbstractItem**
  The currently selected item

Methods

- **public bool filterAcceptsRow(int source_row, QModelIndex source_parent)**
  Tells by analysing the given row if it should be shown or not. This behaviour can be modified via setFilterRegExp method so that e.g. only the entries of a specific host can be shown.

- **public bool lessThan(QModelIndex left, QModelIndex right)**
  Defines the sorting behaviour when comparing two entries of model item by telling how to compare these.
- **public void filter_by_item(AbstractItem item)**
  Used to tell the filter by which item it should filter. If the AbstractItem is None all log entries should be shown.

2.5.13 SizeDelegate: QtGui.QStyledItemDelegate



Makes it possible to change the font size of the Gui-Plugin content

Figure 2.40: The ROSModel

Attributes

- **private int current_font_size**
  The size displayed font

Methods

- **public void paint(QPainter painter, QStyleOptionViewItem option, QModelIndex index)**
  Defines how the items of the model will be painted in the view. Can be used to draw e.g. bigger or smaller fonts.
- **public void set_bigger_font_size()**
  Increases the displayed font-size
- **public void set_smaller_font_size()**
  Decreases the displayed font-size

2.6 GUI - View

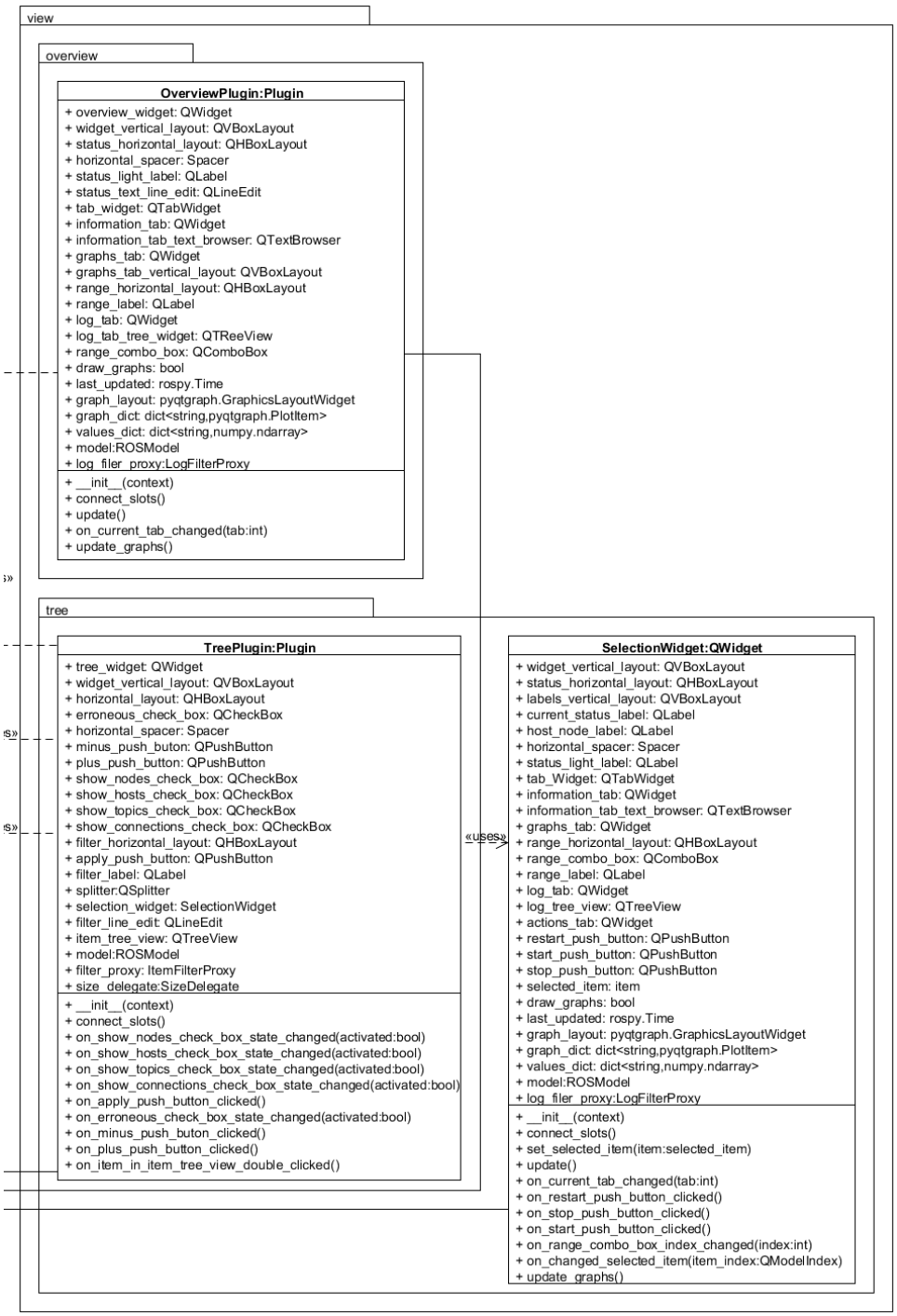
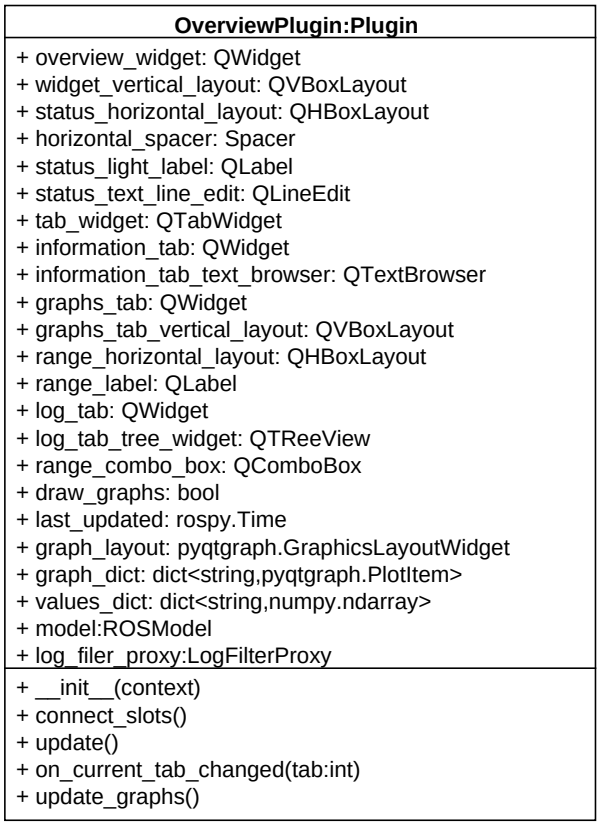


Figure 2.41: The view class diagram

2.6.1 OverviewPlugin



The OverviewPlugin is the core of the graphical user interface, which contains most of the relevant information in a small and fancy area.

Figure 2.42: The ROSModel

Attributes

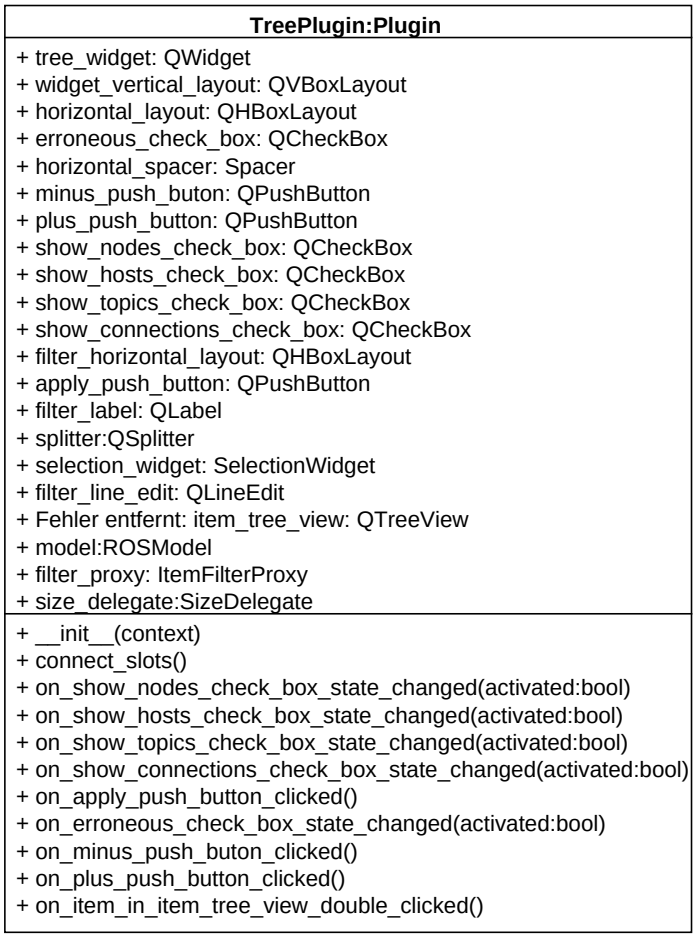
- **public QWidget overview_widget**
  The object which holds the widget
- **public QLabel status_light_label**
  A status ligth wich shows if everything is ok or not
- **public QTabWidget tab_widget**
  The object wich holds the different tabs of the widget
- **public QWidget information_tab**
  A tab wich gives general information about the network
- **public QWidget graphs_tab**
  Displays graphs about the network

- **public QComboBox range_combo_box**
  Makes it possible to set the range of the graphs
- **public QWidget log_tab**
  Shows actual errors and warnings
- **public bool draw_graphs**
  When the graph tab is selected, draw_graphs is set on true an the graph will appear
- **public rospy.Time last_update**
  The time of the latest update
- **public pyqtgraph.GraphicsLayoutWidget graph_layout**
  The layout where the graphs will be plotted. Graphs are modelled as PlotItems.
- **public dict{string, pyqtgraph.PlotItem} graph_dict**
  Dictionary of the names of the values together with the graphs represented as PlotItems.
- **public dict{string, numpy.ndarray} values_dict**
  Dictionary of the names of the values together with the values as an array for fast plotting
- **public ROSModel model**
  The model used to show the content
- **public LogFilterProxy log_fiter_proxy**
  The LogFilterProxy which is currently used for sorting the logs.

Methods

- **public void connect_slots()**
  Initializes the slots of the widget
- **public void update()**
  Updates the widget and draws the graphs if draw_graphs is true.
- **public void on_current_tab_changed(int tab)**
  The widget wants to get notified when the tab changed so it can e.g. draw the graphs etc.
- **public void update_graphs()**
  Updates and redraws the graphs

2.6.2 TreePlugin



TreePlugin is very simply and shows only the actual active hosts and nodes. It is possible to filter the output, e.g. only erroneus hosts or nodes are displayed.

Figure 2.43: The ROSModel

Attributes

- **public QWidget tree_widget**
  The object wich holds the widget
- **public QCheckBox erroneous_check_box**
  Only erroneous hosts and nodes will be displayed
- **public QCheckBox show_node_check_box**
  Displays the activ nodes
- **public QCheckBox show_host_check_box**
  Displays the activ hosts

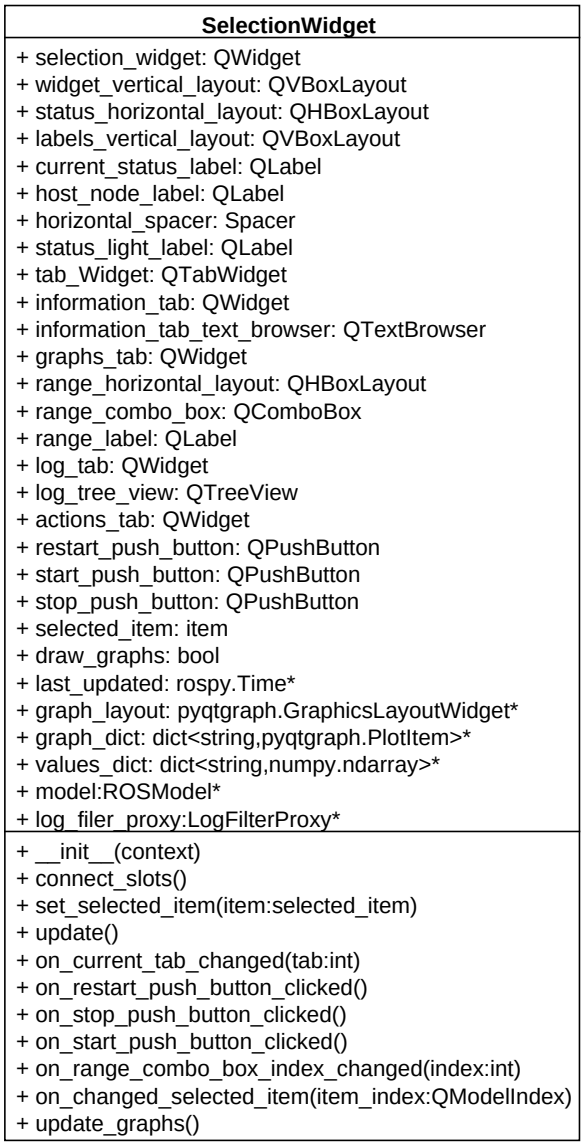
- **public QCheckBox show_topics_check_box**
  Displays the actual topics
- **pubic QCheckBox show_connects_check_box**
  Displays the actual connections
- **public QPushButton plus_push_button**
  Makes it for, a better clarity, possible to zoom in
- **public QPushButton minus_push_button**
  and zoom out
- **public SelectionWidget selection_widget**
  The SelectionWidget which opens on double-click on the TreeView
- **public QLineEdit filter_line_edit**
  A textfield where you can define a filter for the output
- **public ROSModel model**
  The connection to the ROSModel
- **public ItemFilterProxy filter_proxy**
  The filter which will be used to filter the items of the TreeViews e.g. by a search text or according to other criteria selectable by CheckBoxes like show_hosts.
- **private size_delegate: SizeDelegte**
  The size_delegate is responsible for painting the rows in a specific manner e.g. to make the font bigger.

Methods

- **public void connect_slots()**
  Initializes the slots from the widget
- **public void on_show_nodes_check_box_state_changed(bool activated)**
  Displays or delete the nodes in the box wether the check box is set or unset
- **public void on_show_hosts_check_box_state_changed(bool activated)**
  Displays or delete the host in the box wether the check box is set or unset
- **public void on_show_topics_check_box_state_changed(bool activated)**
  Displays or delete the topics in the box wether the check box is set or unset
- **public void on_show_connections_check_box_state_changed(bool activated)**
  Displays or delete the connections in the box wether the check box is set or unset

- **public void on_apply_push_button_clicked()**
  Filters the content in the box according to the content of the filter_line_edit
- **public void on_erroneus_check_box_state_changed()**
  If this check box is set, only erroneus hosts and nodes will be displayed
- **public void on_plus_push_button_clicked()**
  Checks if the plus_push_button is clicked and zoomes in (increases the size of the font)
- **public void on_minus_push_button_clicked()**
  Checks if the minus_push_button is clicked and zoomes out (decreases the size of the font)
- **public void on_item_in_tree_view_double_clicked()**
  Handels the double-click action and opens the clicked item in the SelectionWidget

2.6.3 SelectionWidget



This Widget shows detailed information the currently selected item which might be a host, a node, a topic or a connection.

Figure 2.44: The ROSModel

Attributes

- **public QLabel host_node_label**
  The name of the actual selected item
- **public QLabel status_light_label**
  A status-light about the status of the current item

- **public QTabWidget tab_widget**
  The object wich holds the different tabs of the widget
- **public QWidget information_tab**
  A tab wich gives general information about hosts or nodes
- **public QWidget graphs_tab**
  Displays graphs about the actual selected item, e.g the Network- and CPU-Load
- **public QComboBox range_combo_box**
  Makes it possible to set the range of the graphs
- **public QWidget log_tab**
  Shows actual errors and warnings
- **public QWidget actions_tab**
  Includes buttons to restart and stop nodes
- **public item selected_item**
  The selected item
- **public bool draw_graphs**
  When the graph tab is selected, draw_graphs is set on true an the graph will appear
- **public rospy.Time last_updated**
  The time of the last update
- **public dict{string, pyqtgraph.PlotItem} graph_dict**
  Dict of the names of the values together with the graphs represented as PlotItems.
- **public dict{string, numpy.ndarray} values_dict**
  Dictionary of the names of the values together with the values as an array for fast plotting
- **public ROSModel model**
  The model used to show the content
- **public LogFilterProxy log_filter_proxy**
  The filterproxy which will be used to show only the entries of the current item in the log_tab

Methods

- **public void connect_slots()**
  Initializes the slots from the widget

- **public void set_selected_item(item selected_item)**
  Set the selected item
- **public void update()**
  Updates the widget
- **public void on_current_tab_changed(int tab)**
  Will be called when you switch between tabs
- **public void on_restart_push_button_clicked()**
  Handels the retart button and restarts a host or node
- **public void on_stop_push_button_clicked()**
  Handels the stop button and stops a host or node
- **public void on_start_push_button_clicked()**
  Handels the start button and starts ahost or node
- **public void on_range_combo_box_index_changed(int index)**
  Handels the change of the graph range
- **public void on_changed_selected_item(QModelIndex index)**
  Index handels the change of the selected item
- **public void update_graphs()**
  Updates the graph plot

3 Messagetypes

3.1 HostStatistics

```
# ip of the host
string host

# the statistics apply to this time window
time window_start
time window_stop

# cpu
float32 cpu_temp_mean
float32 cpu_temp_stddev
float32 cpu_temp_max

float32 cpu_usage_mean
float32 cpu_usage_stddev
float32 cpu_usage_max

float32[] cpu_usage_core_mean
float32[] cpu_usage_core_stddev
float32[] cpu_usage_core_max

float32[] cpu_temp_core
float32[] cpu_temp_core_mean
float32[] cpu_temp_core_stddev
float32[] cpu_temp_core_max


# gpu
float32[] gpu_temp_mean
float32[] gpu_temp_stddev
float32[] gpu_temp_max

float32[] gpu_usage_mean
float32[] gpu_usage_stddev
float32[] gpu_usage_max
```

```
# ram
float32 ram_usage_mean
float32 ram_usage_stddev
float32 ram_usage_max

# network

string[] interface_name
int32[] message_frequency_mean
int32[] message_frequency_stddev
int32[] message_frequency_max

# bandwith of each network interface

int32[] bandwith_mean
int32[] bandwith_stddev
int32[] bandwith_max

# drive
string[] drive_name
int32[] drive_free_space

# input output operations on a specific drive
int32[] drive_read
int32[] drive_write
```

3.2 NodeStatistics

```
#ip of the host this node belongs to
string host

#identifier of this node
string node

# the statistics apply to this time window
time window_start
time window_stop

#CPU

float32 node_cpu_usage_mean
float32 node_cpu_usage_stddev
float32 node_cpu_usage_max

float32[] node_cpu_usage_core_mean
float32[] node_cpu_usage_core_stddev
float32[] node_cpu_usage_core_max


#GPU

float32[] node_gpu_usage_mean
float32[] node_gpu_usage_stddev
float32[] node_gpu_usage_max

# ram
float32 node_ramusage_mean
float32 node_ramusage_stddev
float32 node_ramusage_max

# network load of the node

int32 node_message_frequency_mean
int32 node_message_frequency_stddev
int32 node_message_frequency_max
```

```
int32 node_bandwith_mean
int32 node_bandwith_stddev
int32 node_bandwith_max

# Drive I/O statistics of the node

int32 node_write_mean
int32 node_write_stddev
int32 node_write_max

int32 node_read_mean
int32 node_read_stddev
int32 node_read_max
```

3.3 RatedStatistics

```
# name of node/host/connection
string seuid

# the rated statistics apply to statistics from this time window
time window_start
time window_stop

# an array of rated entities
RatedStatisticsEntity[] rated_statistics_entity

# state of the metadata from the node/host/connection :
# state: { 0 = high ; 1 = low ; 2 = unknown}
uint8[] state
```

3.4 RatedStatisticsEntity

```
# type of statistic like cpu_usage_core or cpu_usage
string statistic_type

# the value of the type
string[] actual_value

# the expected value like "40 - 70"
string[] expected_value

# state of the metadata from the node/host/connection :
# state: { 0 = high ; 1 = low ; 2 = unknown}
uint8[] state
```

4 Servicetypes

4.1 NodeReaction

```
# ip of the affected host
string host

# identifier of the affected node
string node

# action  [restart, stop, command]
string action

# command (for action command)
string command
---
# message returned upon completion
string returnmessage
```

4.2 StatisticHistory

```
# get all statistics where timestamp in ms is > than timestamp
time timestamp
---
# each statistic has its own rated statistic.
# timestamps can be found at the time windows of each statistic

HostStatistics[] host_statistics
RatedStatistics[] rated_host_statistics

NodeStatistics[] node_statistics
RatedStatistics[] rated_node_statistics

TopicStatistics[] topic_statistics
RatedStatistics[] rated_topic_statistics
```

5 Sequence diagrams

5.1 Data Acquisition

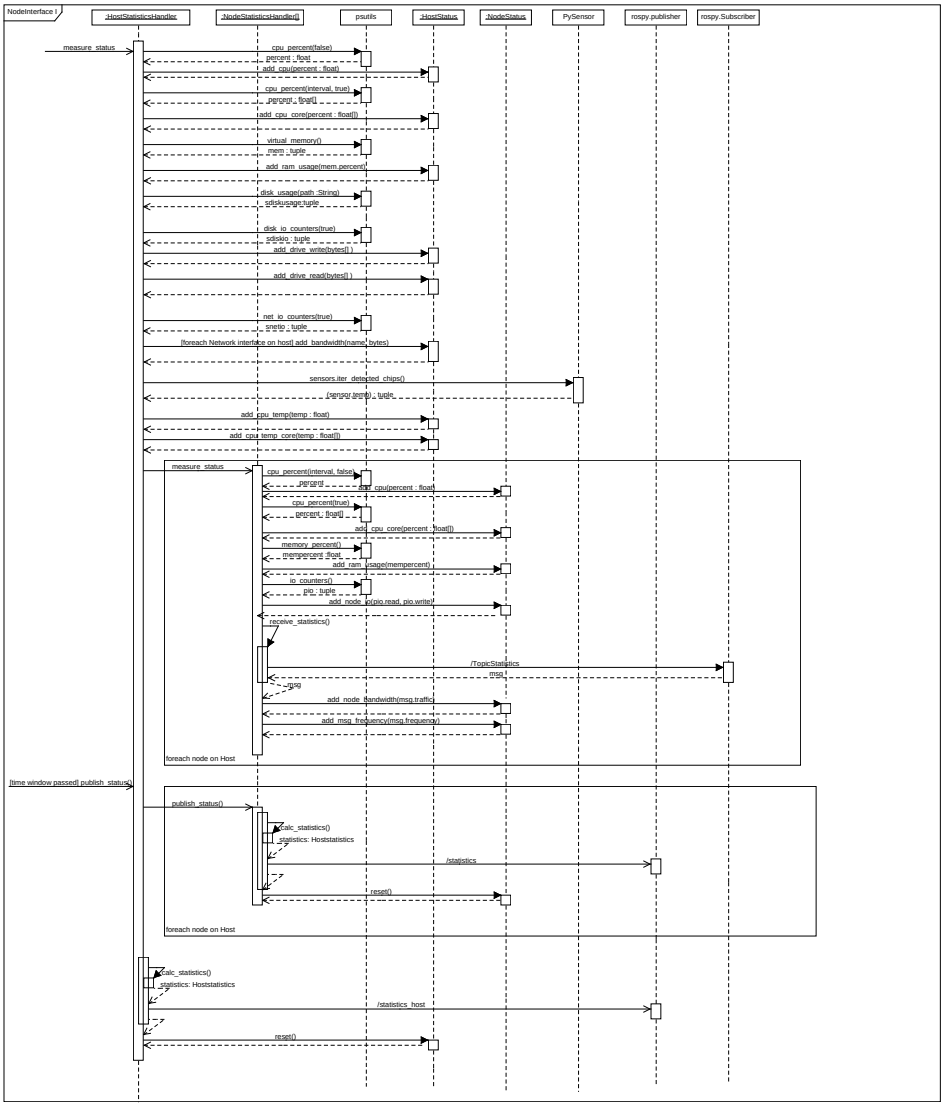


Figure 5.1: Used psutils calls to collect data and publishing to topics.

Acquisition

The Acquisition is triggered by a timer. The HostStatisticsHandler uses psutils and PySensors to collect data about it's current state and write it into the HostStatus instance. It also triggers

an asynchronous call of measure_status to all of it's node's executed in individual threads. The nodes use psutils and ROS Topic Statistics to collect data and write it into the NodeStatus instance.

Publishing

After the time window has passed the publishing process is triggered. The HostStatisticsHandler again triggers asynchronous calls to all of it's nodes to publish their status. During the publishing process, the statistical values are calculated and transformed into a ROS message. After publishing the data, the status of the host or node is resetted.

5.2 Dataprocessing and -storage

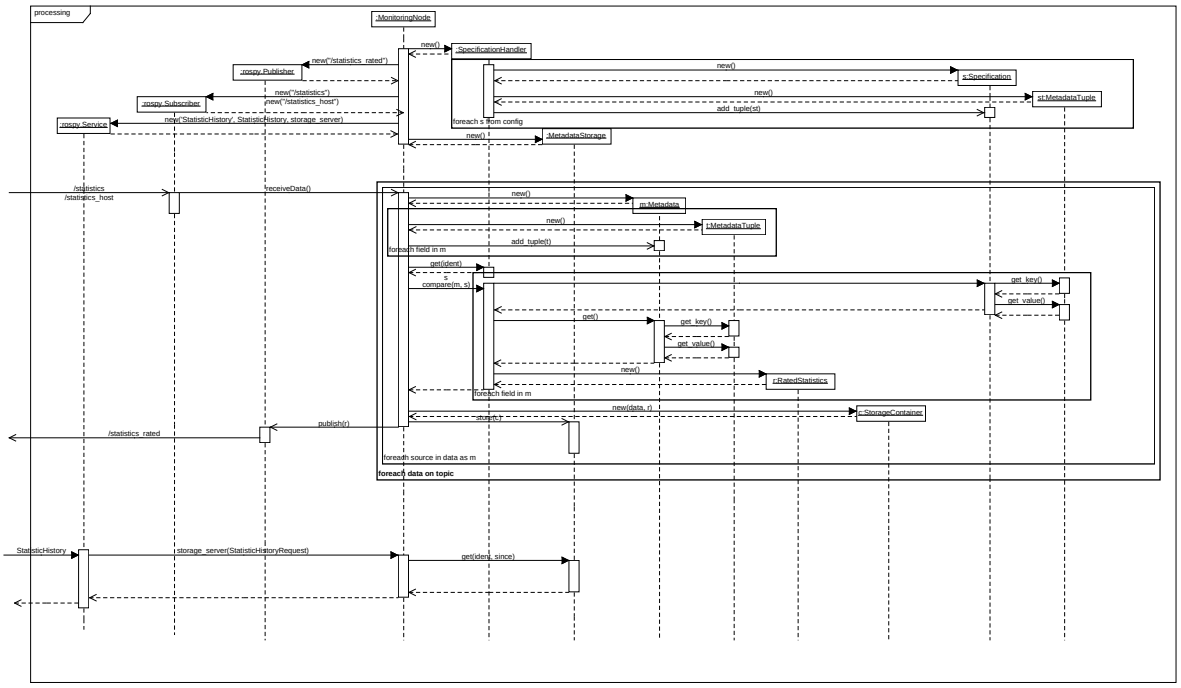


Figure 5.2: Three sequences appearing in the data-processing part of the project.

Setup

During the first activity of the MonitoringNode, it sets up a SpecificationHandler, which then loads all specifications from config files and stores them in MetadataTuple objects bundled in Specification objects.
It then sets up the MetadataStorage.

Receiving data

The second activity of the MonitoringNode is triggered on receiving data on either the /statistics or /statistics_host topic. The incoming data is translated into Metadata objects containing of several MetadataTuples describing every measurement featured in the received data.
Now the MonitoringNode looks up a Specification from SpecificationHandler concerning the connection/node/host it just received data about.
On success it compares the created Metadata object with the found Specification object MetadataTuple-wise for each field featured in the Metadata/Specification object.
Erroneous results will be marked in a new RatedStatistics object. Bundled with the raw input

data, a timestamp and an identifier describing the concerned connection/node/host it will be stored in the MetadataStorage object created on setup.

Providing data for the GUI

Answering a request for all data or a special identifier describing a connection/node/host since a given point of time, the MonitoringNode will return the matching data from the MetadataStorage. A result of that will contain raw data, rated data, a timestamp and the identifier mentioned above.

5.3 Countermeasures

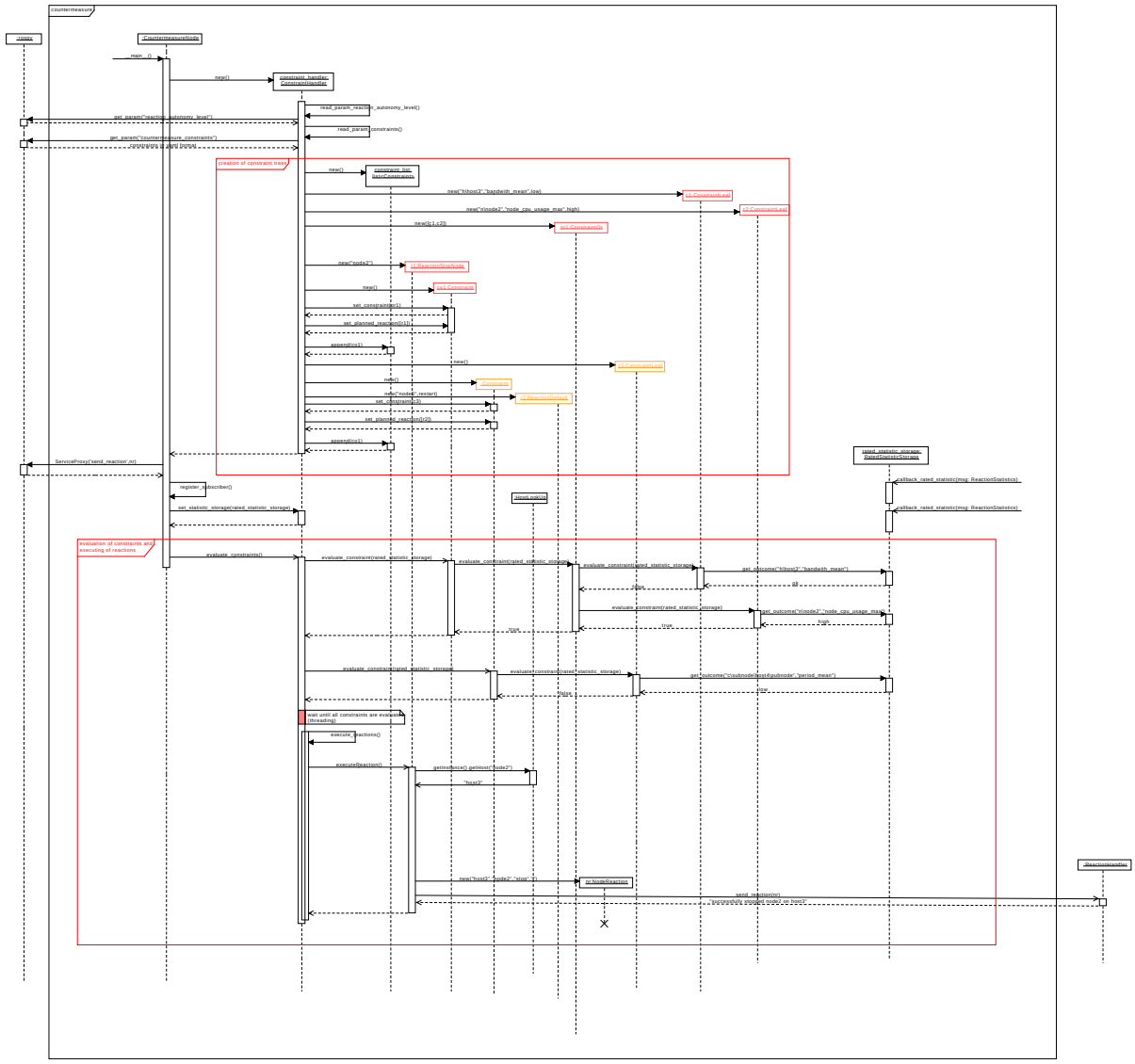


Figure 5.3: Initialisation of Constraints and evaluating and executing countermeasures.

This sequence diagram shows how constraints get initialised from the parameter server. Additionally it states how an constraint first gets evaluated periodically and then executes it's reaction when necessary.

Initialisation of Constraints

First the ConstraintHandler asks for all constraints from the parameter server. Then the class parses them into Constraints, building an tree for efficient evaluation. Each constraint has its

own array of reactions to take when necessary.

Evaluation and reacting

The CountermeasureNode regularily asks the ConstraintHandler to evaluate the constraints again. Each evaluate call in an ConstraintItem gets passed on to the objects branches until the leafes are reached. Each leaf then evaluates if the specific contraint for one specific statistic datapoint is true and passes its result on back on its way to the root of the tree.
After the evaluation the ConstraintHandler checks if there is the need to react. The reaction creates a new NodeReaction object and sends it to the remote ReactionHandler.

5.4 GUI

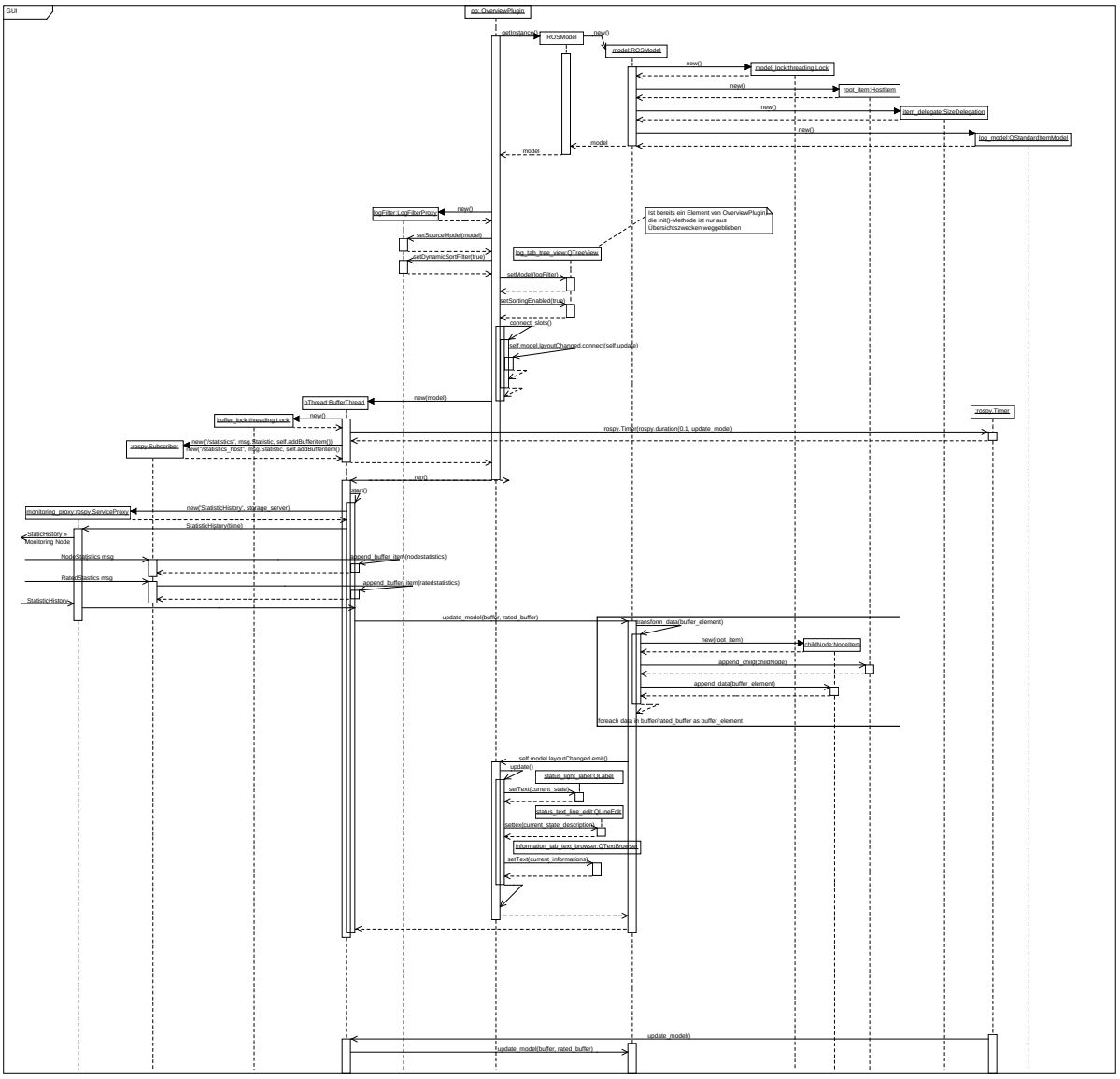


Figure 5.4: A example GUI sequence.

Setup

In this sequence diagramm it is exemplary shown how the GUI part of the Software initializes. All widgets do whenever startet proceed the following steps: Initialize their own data, create or get the model, synchronize with an additional proxy model and then create and start the BufferThread so data is flowing in. The BufferThread then connects via the services to the

MonitoringNode and gets a history of the last messages. Finally the widgets connect their slots so that e.g. the view gets updated when the model changes.

Running

When the GUI is running everything is getting updated with Qt's signal and slot mechanism. The BufferThread gets all actual data as a Subscriber of all topics and then regularily redirects this data by updating the model. The changes are then transmitted to the view aka the widgets which then show this incoming data.

Any other information on how the classes work and interact can often be discovered by looking at the class diagram.