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, 07.06.2014

Inhaltsverzeichnis

1 Composition

1.1 Architecture

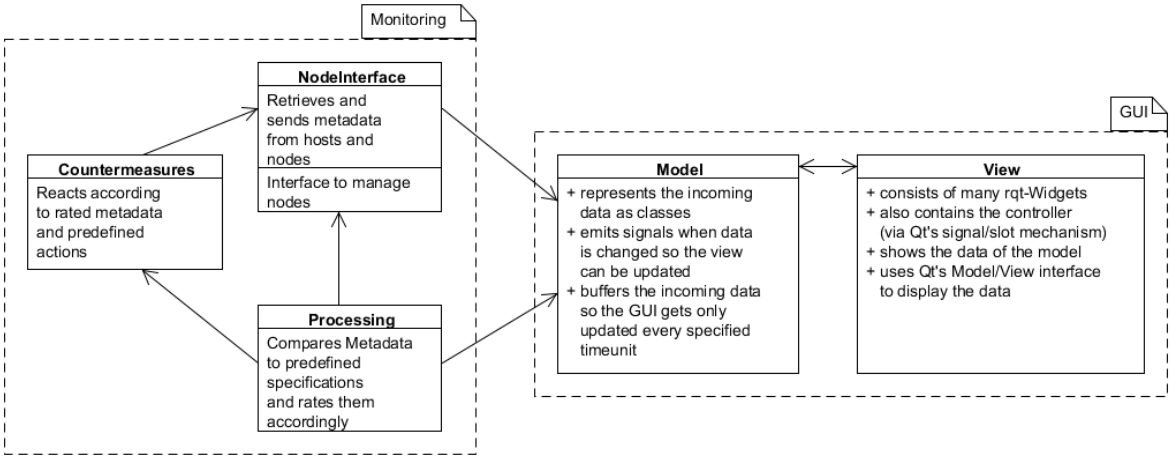


Abbildung 1.1: architecture

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 MVC architecture, consisting of the usual three elements: model, view and controller. 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

2 Classes Description

2.1 Processing

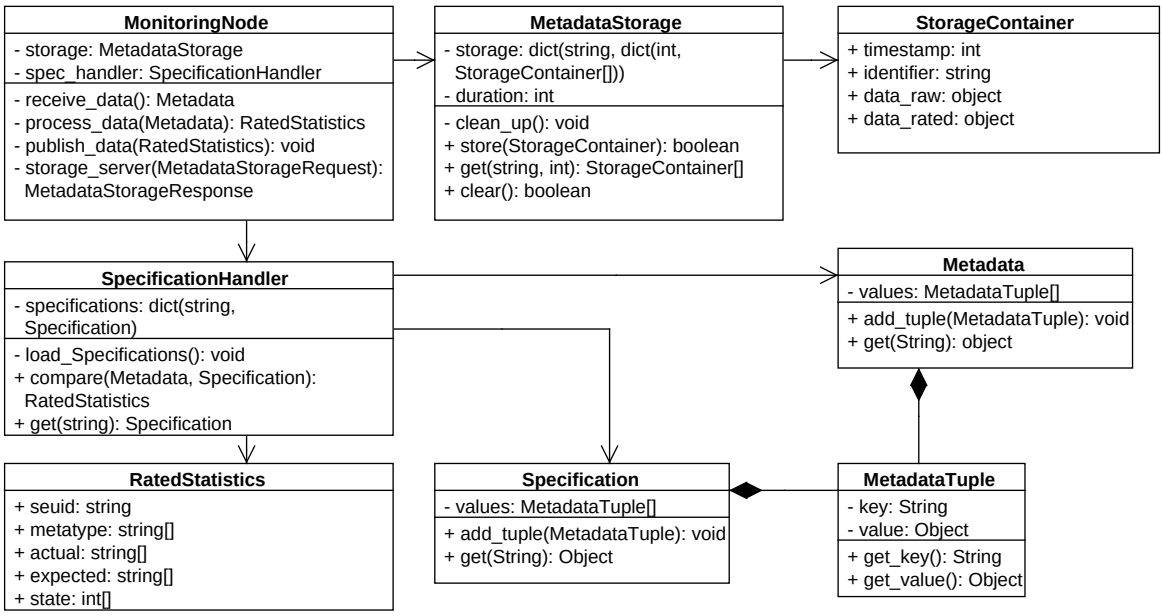


Abbildung 2.1: The UML diagram of the processing package

2.1.1 MonitoringNode

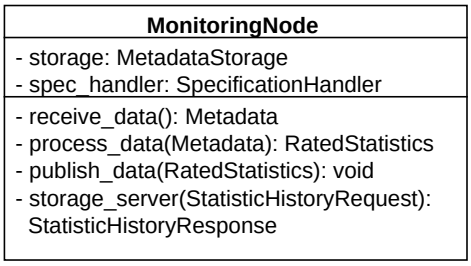


Abbildung 2.2: The MonitoringNode

Main Class wrapping the processing functionality.

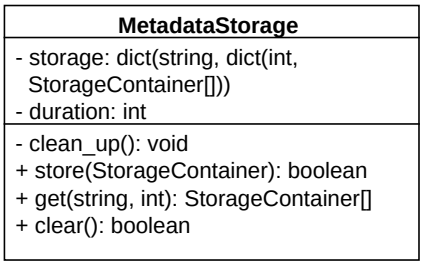
Attributes

- 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)**
  Listen 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.

Abbildung 2.3: The MetadataStorage

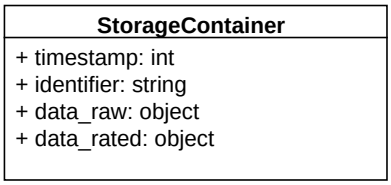
Attributes

- **private dict(string, dict(int, StorageContainer[])) storage**
  Datastructure to store Packages by key and timestamp
- **private int duration**
  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



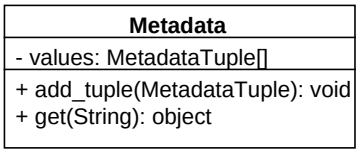
Wraps Metadata in raw and rated form with an identifier and a timestamp. Object to be returned on request by the GUI model.

Abbildung 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.

Abbildung 2.5: The Metadata

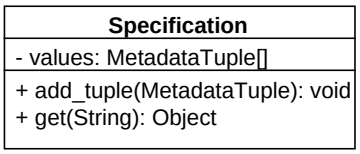
Attributes

- **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.

Abbildung 2.6: The Specification

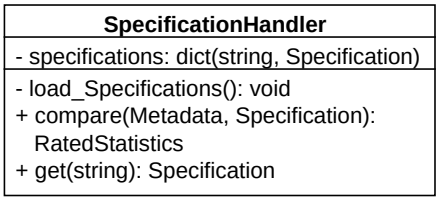
Attributes

- **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.

Abbildung 2.7: The SpecificationHandler

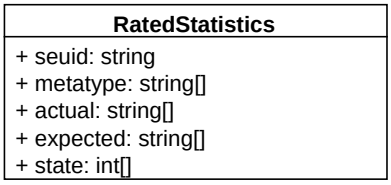
Attributes

- **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



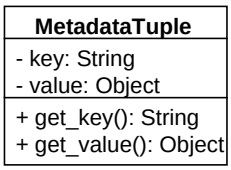
Wraps the result of the comparison between the actual metadata and the specificaton.

Abbildung 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.

Abbildung 2.9: The MetadataTuple

Attributes

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

Methods

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

2.2 NodesInterface

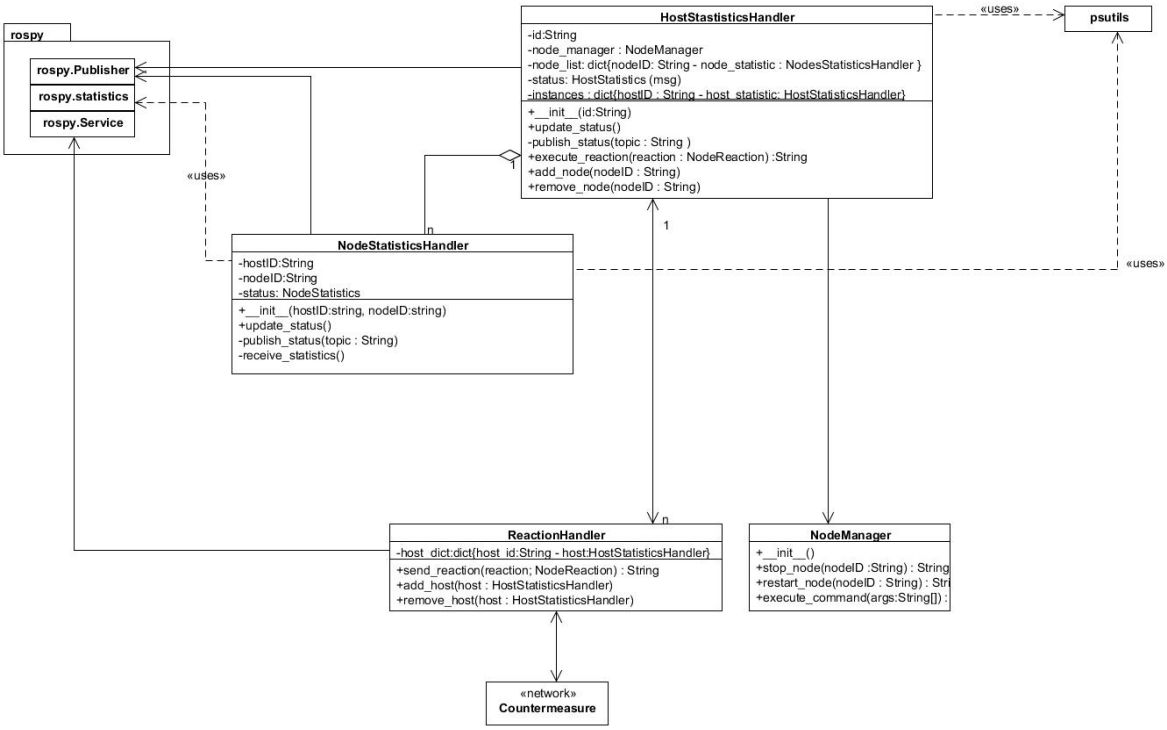


Abbildung 2.10: UML diagram of the NodeInterface package

2.2.1 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 string id**
  ip of the host.
- **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 HostStatistics status**
  Holds the current statistics about the status of 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 update_status()**
  collects statistics about the host's current status using psutils.
- **private void publish_status(String topic)**
  publishes the current status to a topic using ROS's publisher-subscriber mechanism.
- **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.

2.2.2 NodeStatisticsHandler

Holds the statistics of an individual Node.

Attributes

- **private String hostID**
  id of the host this node runs on
- **private String nodeID**
  identifier of this node
- **private NodeStatistics status**
  current statistcs about the status of the node

Methods

- **public void update_status()**
  collects statistics about the node's current status using psutils and rospy.statistics
- **private void publish_status()**
  publishes the current status to a topic using ROS's publisher-subscriber mechanism.
- **private void receive_statistics()**
  receives the statistics published by ROS Topic statistics

2.2.3 NodeManager

can restart or stop nodes or execute a countermeasure

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.

2.2.4 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

2.2.5 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

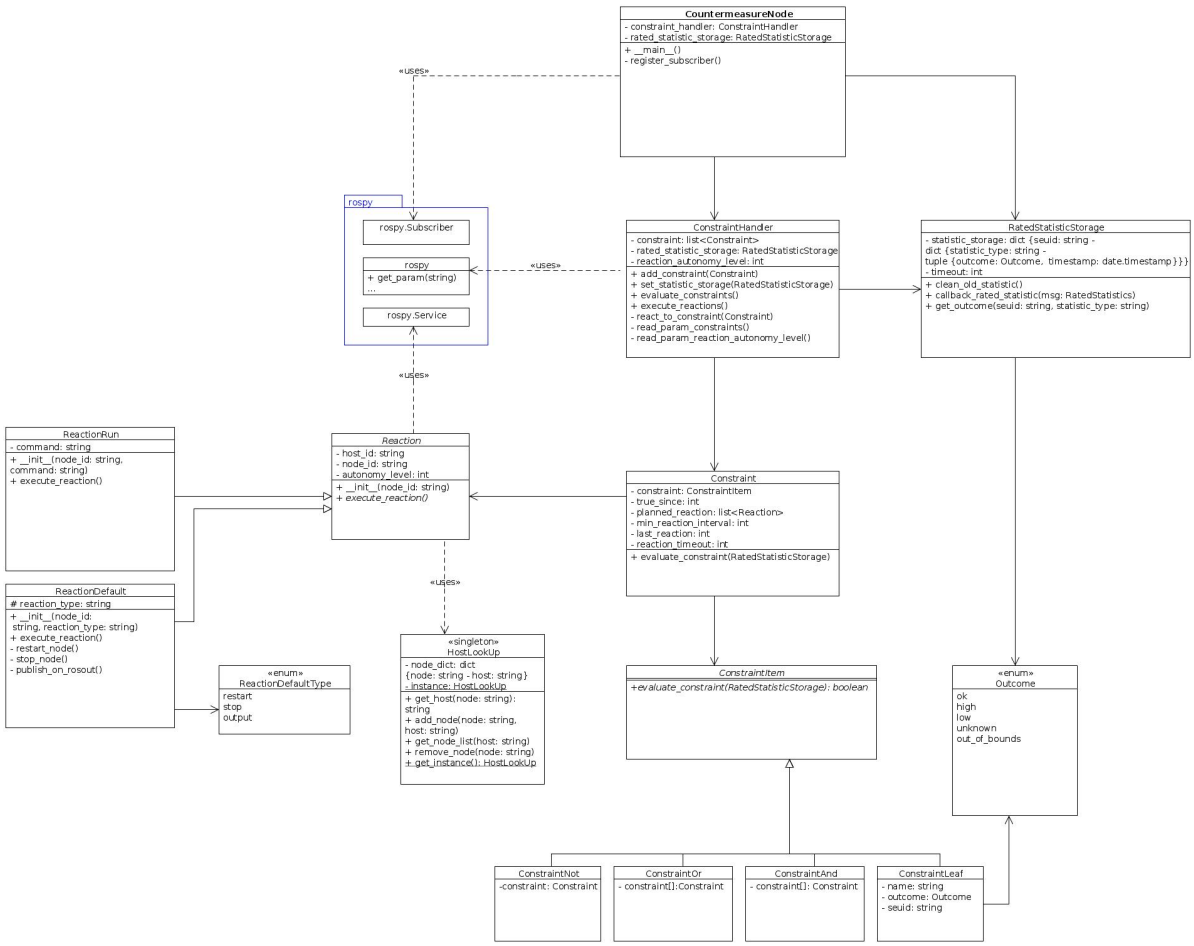


Abbildung 2.11: The UML diagram of the countermeasure package

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.

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 (threads) 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.

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.

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.

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. 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 an logical operation on constraints or an actual constraint

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 constraint.

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

Attributes

- **private Constraint[] constraint**
  contains constraints to be evaluated with a logical and

Methods

- **public boolean evaluate_constraint(RatedStatisticStorage)**
  returns true if the evaluation of both constains returns true

2.3.8 ConstraintOr

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

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

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

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 runs a command as action

Attributes

- **private string command**
  contains the command

Methods

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

2.3.13 ReactionDefault

attributes

- **private ReactionDefaultType reaction_type**
  containts the type this reaction is of.

Methods

- **public void __init__(string node_id, string reactionType)**
  initializes the reaction. sets the reactiontype of this reaction.
- **public void exececute_reaction()**

- **private void restart_node()**
  restarts the Node
- **private void stop_node()**
  stops the Node
- **private void publish_on_rosout()**
  publishes the cause of the reaction on rosout.

2.3.14 Enum ReactionDefaultType

Types

- **restart**
  reaction is a restart of an entity
- **stop**
  reaction is stopping an entity

- **output**
  reaction is publishing the reaction on rosout

2.3.15 HostLookUp

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

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

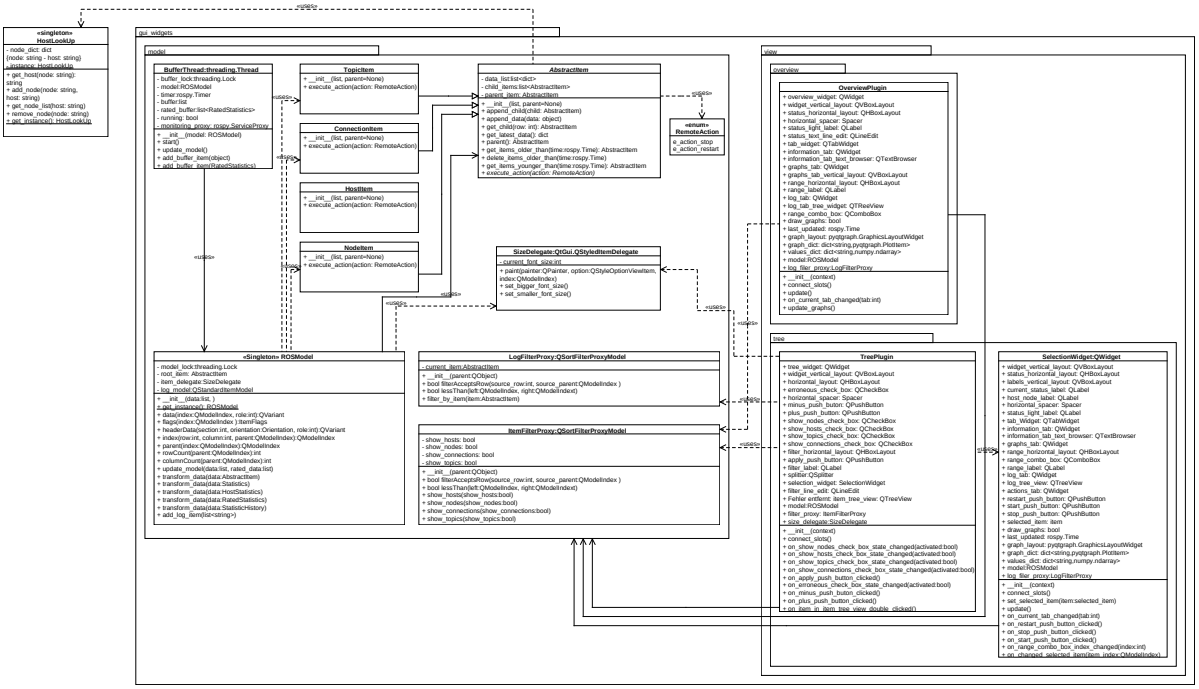


Abbildung 2.12: The GUI class diagram

2.5 GUI - Model

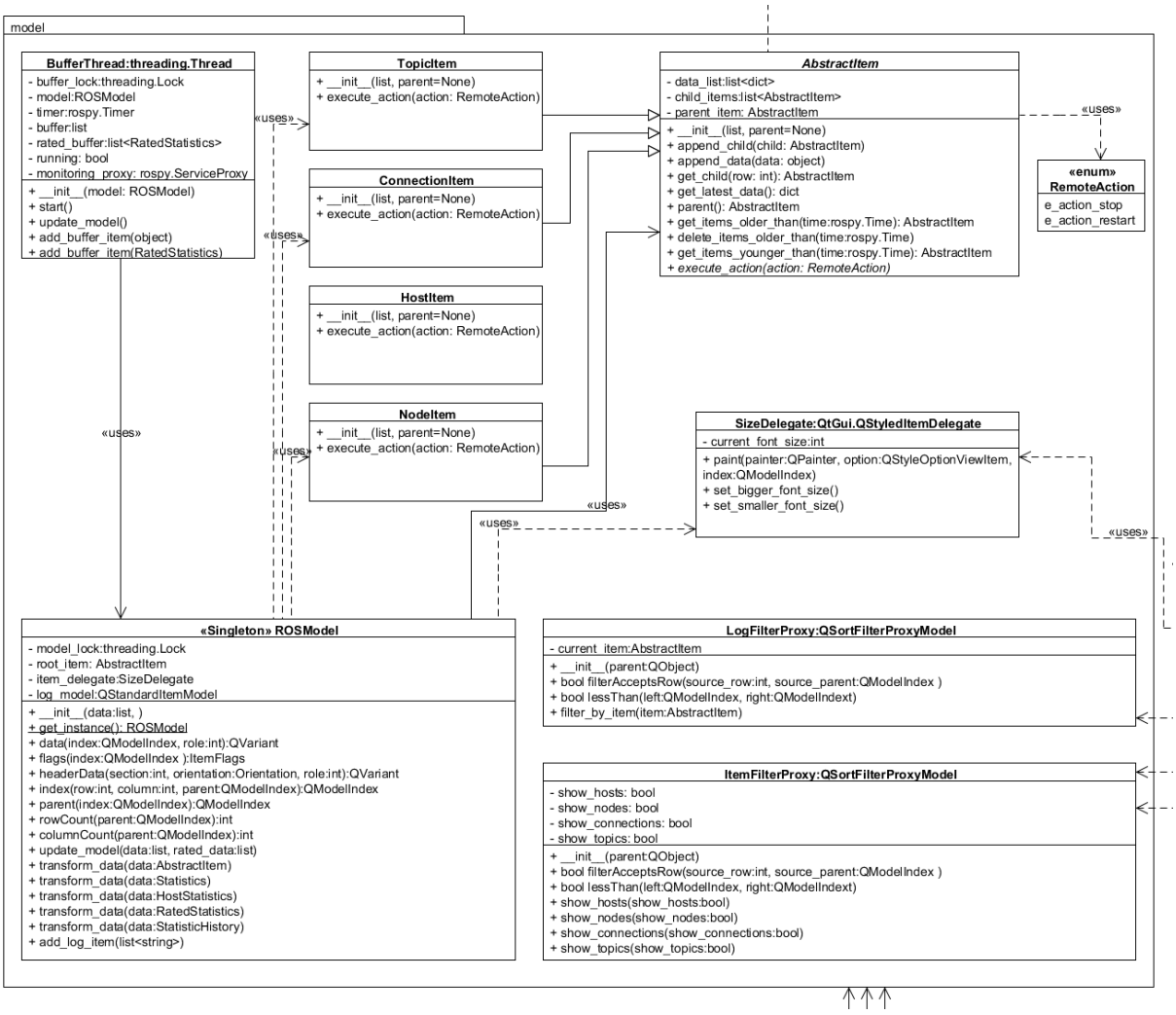


Abbildung 2.13: The model class diagram

2.5.1 BufferThread

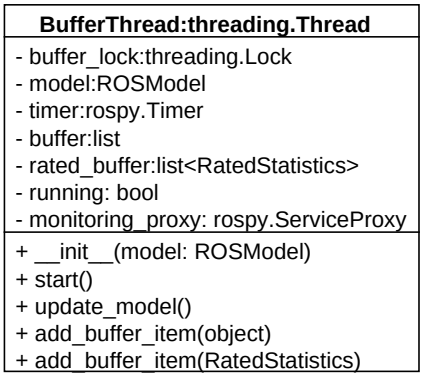


Abbildung 2.14: The BufferThread

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

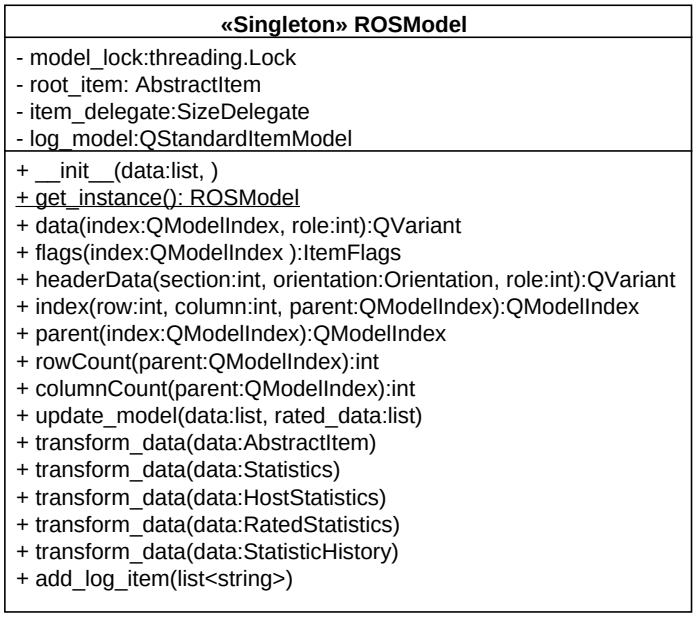
Attributes

- **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 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
- **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.

Abbildung 2.15: The ROSModel

Attributes

- **private threading.Lock model_lock**
  protects the model from parallel modification

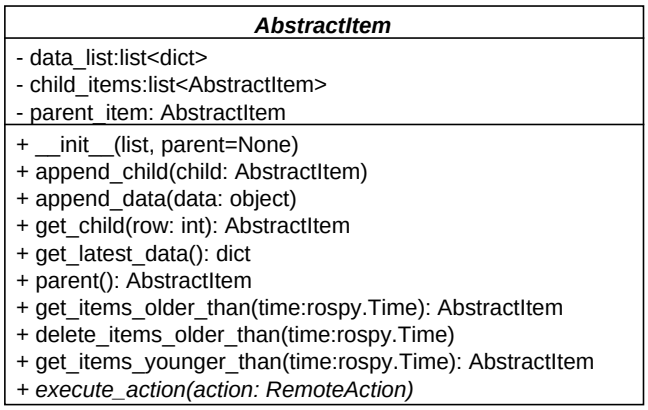
- **private AbstractItem root_item**
  contains the list of headers
- **private SizeDelegate item_delegate**
  the item_delegate is responsible for
- **private QStandardItemModel log_model**

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



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

Abbildung 2.16: The 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

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)**
  return the child at the position row
- **public dict get_latest_data()**
  return 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 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

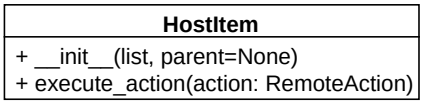


Abbildung 2.17: The HostItem

A HostItem represents a host with all its data. The data_list will contain dicts of entries including

Methods

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

2.5.5 NodeItem

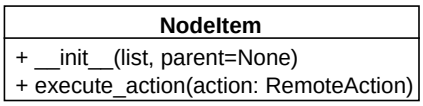


Abbildung 2.18: The NodeItem

A NodeItem represents a node with all of its data. It also has a interface to start/stop/restart nodes. The data_list will contain dicts of entries including

Methods

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

2.5.6 TopicItem

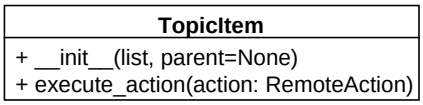


Abbildung 2.19: The TopicItem

A TopicItem reprensents a specific topic which contains many connections and has attributes like the number of sent messages. The data_list will contain dicts of entries including

Methods

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

2.5.7 ConnectionItem

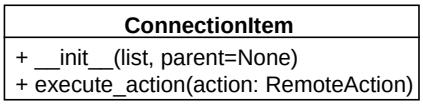


Abbildung 2.20: The ConnectionItem

A ConnectionItem reprensents the connection between a publisher and a subscriber and the topic they are puglishing / listenening on. The data_list will contain dicts of entries including

Methods

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

2.5.8 Enum RemoteAction

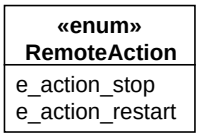


Abbildung 2.21: The ROSModel

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

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

ItemFilterProxy:QSortFilterProxyModel
- show_hosts: bool - show_nodes: bool - show_connections: bool - show_topics: bool
+ __init__(parent:QObject) + bool filterAcceptsRow(source_row:int, source_parent:QModelIndex) + bool lessThan(left:QModelIndex, right:QModelIndext) + show_hosts(show_hosts:bool) + show_nodes(show_nodes:bool) + show_connections(show_connections:bool) + show_topics(show_topics:bool)

Abbildung 2.22: 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 modify 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

LogFilterProxy
- current_item:AbstractItem
+ __init__(parent:QObject) + bool filterAcceptsRow(sourceRow:bool filterAcceptsRow(source_row:int, source_parent:QModelIndex) + bool lessThan(left:QModelIndex, right:QModelIndext) + filter_by_item(item:AbstractItem)

Abbildung 2.23: 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

SizeDelegate:QtGui.QStyledItemDelegate
- current_font_size:int
+ paint(painter:QPainter, option:QStyleOptionViewItem, index:QModelIndex) + set_bigger_font_size() + set_smaller_font_size()

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

Abbildung 2.24: 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

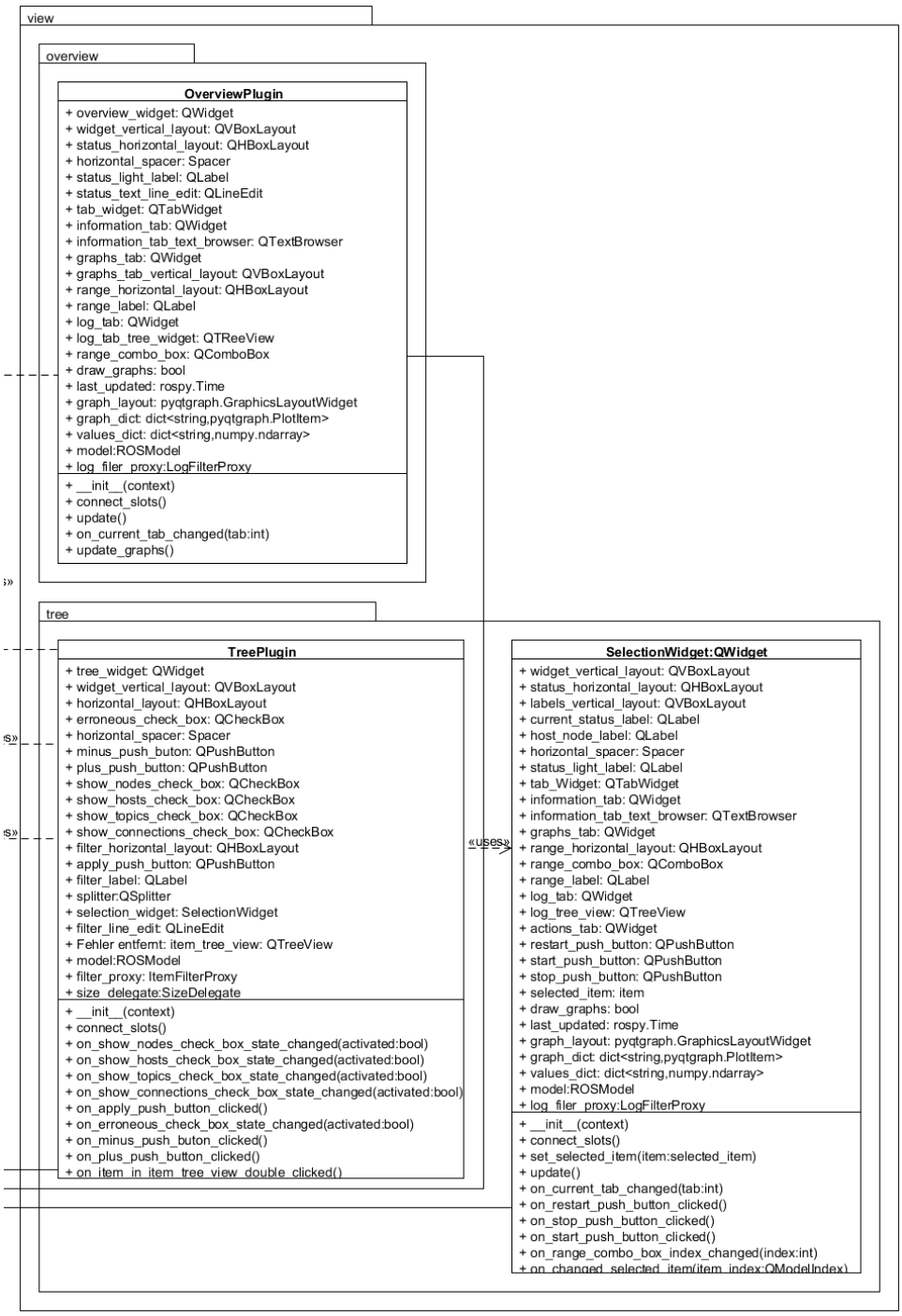


Abbildung 2.25: The view class diagram

2.6.1 OverviewPlugin

OverviewPlugin
+ overview_widget: QWidget + widget_vertical_layout: QVBoxLayout + status_horizontal_layout: QHBoxLayout + horizontal_spacer: Spacer + status_light_label: QLabel + status_text_line_edit: QLineEdit + tab_widget: QTabWidget + information_tab: QWidget + information_tab_text_browser: QTextBrowser + graphs_tab: QWidget + graphs_tab_vertical_layout: QVBoxLayout + range_horizontal_layout: QHBoxLayout + range_label: QLabel + log_tab: QWidget + log_tab_tree_widget: QTReeView + range_combo_box: QComboBox + draw_graphs: bool + last_updated: rospy.Time + graph_layout: pyqtgraph.GraphicsLayoutWidget + graph_dict: dict<string,pyqtgraph.PlotItem> + values_dict: dict<string,numpy.ndarray> + model:ROSModel + log_filer_proxy:LogFilterProxy
+ __init__(context) + connect_slots() + update() + on_current_tab_changed(tab:int) + update_graphs()

Abbildung 2.26: The ROSModel

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

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**
  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_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

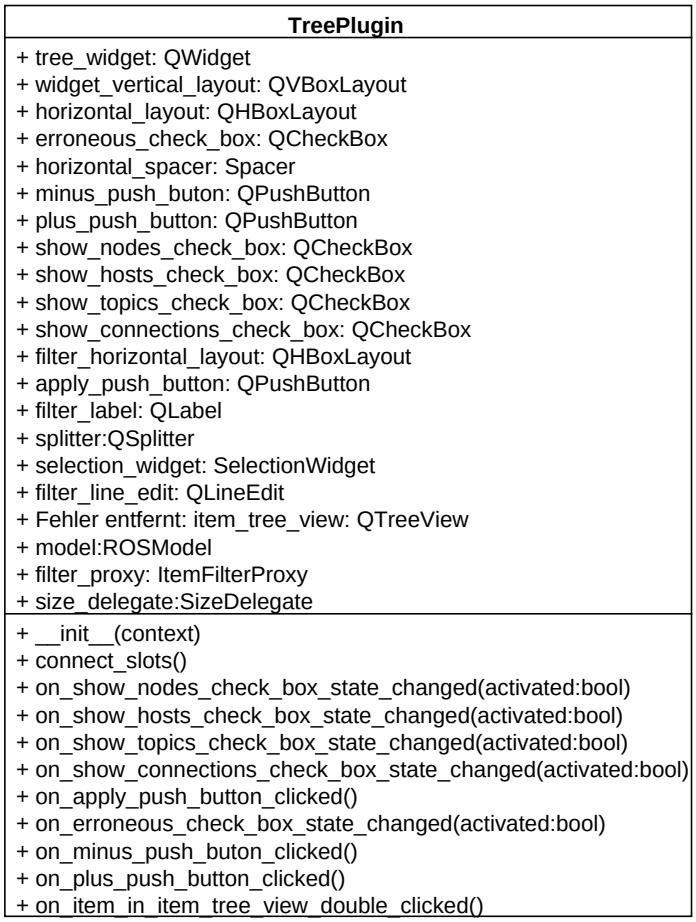


Abbildung 2.27: The ROSModel

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.

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**

- **public size_delegate: SizeDelegte**

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

```
SelectionWidget
+ selection_widget: QWidget
+ widget_vertical_layout: QVBoxLayout
+ status_horizontal_layout: QHBoxLayout
+ labels_vertical_layout: QVBoxLayout
+ current_status_label: QLabel
+ host_node_label: QLabel
+ horizontal_spacer: Spacer
+ status_light_label: QLabel
+ tab_Widget: QTabWidget
+ information_tab: QWidget
+ information_tab_text_browser: QTextBrowser
+ graphs_tab: QWidget
+ range_horizontal_layout: QHBoxLayout
+ range_combo_box: QComboBox
+ range_label: QLabel
+ log_tab: QWidget
+ log_tree_view: QTreeView
+ actions_tab: QWidget
+ restart_push_button: QPushButton
+ start_push_button: QPushButton
+ stop_push_button: QPushButton
+ selected_item: item
+ draw_graphs: bool
+ last_updated: rospy.Time*
+ graph_layout: pyqtgraph.GraphicsLayoutWidget*
+ graph_dict: dict<string,pyqtgraph.PlotItem>*
+ values_dict: dict<string,numpy.ndarray>*
+ model:ROSModel*
+ log_filer_proxy:LogFilterProxy*
---
+ __init__(context)
+ connect_slots()
+ set_selected_item(item:selected_item)
+ update()
+ on_current_tab_changed(tab:int)
+ on_restart_push_button_clicked()
+ on_stop_push_button_clicked()
+ on_start_push_button_clicked()
+ on_range_combo_box_index_changed(index:int)
+ on_changed_selected_item(item_index:QModelIndex)
+ update_graphs()
```

Abbildung 2.28: The ROSModel

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

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)** item_index handels the change of the selected item
- **public void update_graphs()**
  updates the graph plot

3 Sequence diagrams

3.1 Dataprocessing and -storage

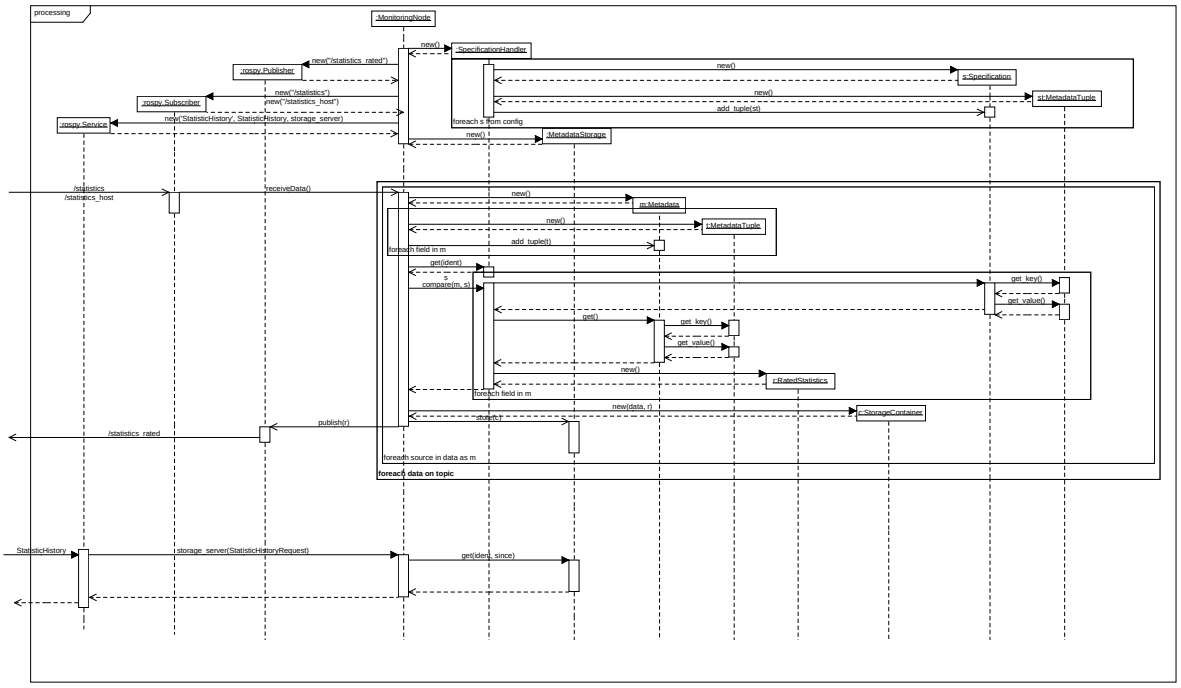


Abbildung 3.1: Three sequences appearing in the data-processing part of the project.

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.