RobMoSys-SROC

MIRoN

QoS Metrics-In-the-loop for
better ROBot Navigation

THE MIRoN FRAMEWORK

THIS PROJECT HAS RECEIVED FUNDING FROM THE *EUROPEAN UNION'S HORIZON 2020 RESEARCH AND INNOVATION PROGRAMME* UNDER GRANT AGREEMENT NO. 732410, IN THE FORM OF FINANCIAL SUPPORT TO THIRD PARTIES OF THE ROBMOSYS PROJECT

Content

1 Overview

One of the main technical objectives of the MIRoN ITP [1] were to develop the MIRoN Framework, including the MIRoN metamodel, one or more textual model editors and code generators, and the required software infrastructure needed to run the generated MIRoN Adaptation Engine within a SmartSoft-based robotics solution. The development of the MIRoN Framework required, as a first step, to appropriately align MOOD2BE [2,3] and RoQME [4,5] (previous ITPs, both funded within the RobMoSys Call 1) so that their metamodels and associated tools were conveniently adapted/extended to be readily usable in MIRoN. Besides, it was also essential to analyse potential adaptation scenarios in order to identify the most relevant modelling primitives to be supported by the MIRoN modelling tools so that the resulting MIRoN models were both simple and expressive.

The MIRoN framework has been developed as a set of Eclipse plug-ins and additional software tools, ready to be installed and used within the SmartMDSD Toolchain [6]: one of the RobMoSys baseline frameworks for robotics software development. The MIRoN framework provides designers with: (i) a textual model editor allowing them to specify variation points in the robot behaviour and define how these variation points should be configured at runtime according to the perceived situation; and (ii) a generator that, taking the previous models as an input, generates and appropriately configures the runtime infrastructure needed to monitor relevant non-functional properties and, according to their evolution, perform the appropriate behaviour adaptations to meet the required robot quality-of-service (QoS).

2 Metamodels and textual model editors included in the framework

The following sections describe the languages that are part of the MIRoN framework. We will start with the tools for the specification of a Behaviour Tree.

2.1 Behaviour Tree

We have developed an EMF-based metamodel for specifying Behaviour Tree (BT) models. This metamodel includes all the modelling concepts originally covered in MOOD2BE, plus those required by MIRoN for specifying decision points (namely, *Switch* and *VariantAction* nodes). Associated with this metamodel, a textual model editor, developed using Xtext [7], now allows Behaviour Designers to create XMI-based BT models following the XML-based syntax required by the MOOD2BE BT interpreter. Figure 1 shows an example model in the BT editor developed for MIRoN ITP. Regarding the abstract syntax of the language, that is, the definition of the concepts that make up the language and their relationships, it is defined by the metamodel shown in Figure 2. The concepts included in the metamodel are described next.

intralogistics.bt

```
<?xml version="1.0"?>
<root main_tree_to_execute="Main">

    <BehaviorTree ID="Main">
        <SubTreePlus ID="PickDeliverCharge" name="PickDeliverChargeSubtree"/>
    </BehaviorTree>

    <BehaviorTree ID="Charge">
        <Sequence name="Charge">
            <SubTreePlus ID="GoTo" __shared_blackboard="false" x_input="-9.33" y_input="2.17"/>
        </Sequence>
    </BehaviorTree>

    <BehaviorTree ID="Deliver">
        <Sequence name="deliver">
            <SubTreePlus ID="GoTo" __shared_blackboard="false" x_input="-8.08" y_input="4.11"/>
            <Action ID="objectdrop" obj_index="-1"/>
        </Sequence>
    </BehaviorTree>
```

Figure 1. Example model in the BT editor developed for the MIRoN project

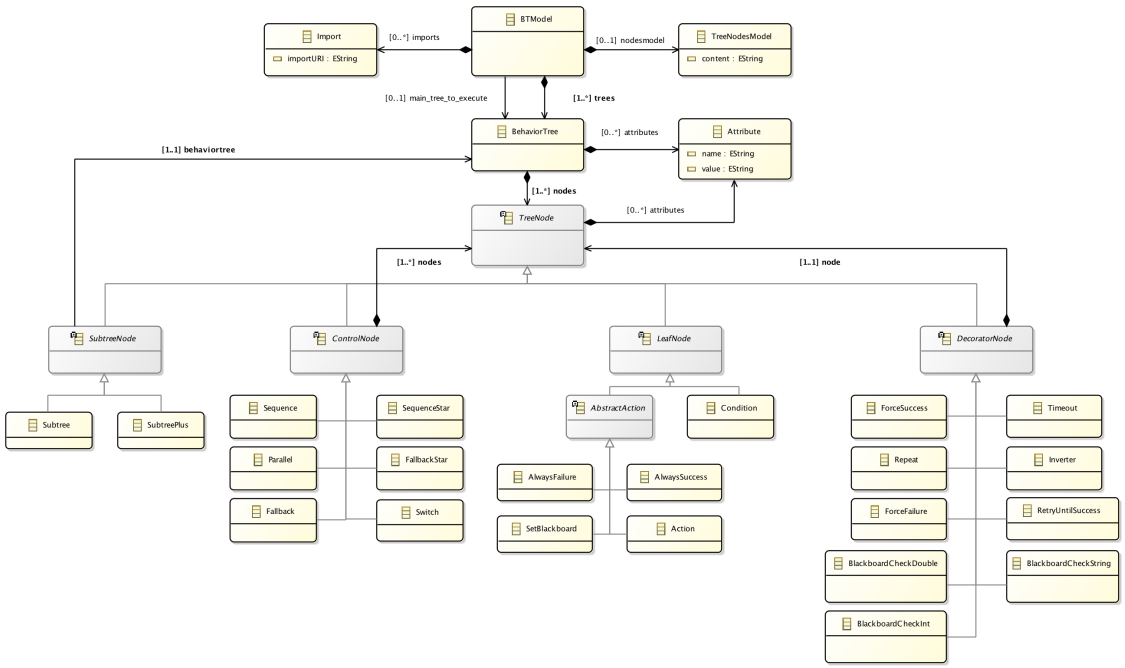

Figure 2. Behaviour Tree Metamodel

BTModel	
Description	Root element of the XML BT model.
Attributes	***main_tree_to_execute:*** name of the main BT that will be executed.
References	***imports***: references to other XML models that need to be loaded. Importing a model makes its elements accessible to be referenced from the current model. ***trees:*** one or more BehaviorTree elements. ***nodesmodel***: optional node definitions through a TreeNodesModel element.

Import	
Description	XML BT model to be imported.
Attributes	***ImportURI:*** URI of the XML BT model to be imported.

TreeNodesModel	
Description	Provides node definitions.
Attributes	***content***: specification treated as opaque metadata whose syntax is not checked.

BehaviourTree	
Description	Representation of a BT.
References	***attributes***: pairs name-value representing the possible attributes that contains the BehaviorTree node. ***nodes:*** collection of nodes contained by the BT.

TreeNode	
Description	Abstract representation of a BT node. Available subclasses are as follows: ***SubtreeNode*** ***ControlNode*** ***LeafNode*** ***DecoratorNode***

SubtreeNode	
Description	Abstract representation of a subtree. Available subclasses are as follows: ***Subtree*** ***SubtreeWrapper***
Super Type	TreeNode
References	***BehaviourTree***: may a contain a reference from the BehaviorTree in another file.

Subtree	
Description	Subtree node does not have any children and her aim is to reference other BehaviorTree situated in another place (XML or Branch).
Super Type	SubtreeNode

SubtreePlus	
Description	Same as SubTree but the it includes support for port remapping.
Super Type	SubtreeNode

ControlNode	
Description	Abstract representation of a control node. The available subclasses are: ***Sequence*** ***SequenceStar*** ***Parallel*** ***Fallback*** ***FallbackStar*** ***Switch***
Super Type	TreeNode
References	***nodes***: collection of nodes contained in (*affected by*) a control node.

Sequence	
Description	Representation of a sequence control node. A Sequence ticks all its children as long as they return SUCCESS. If any child returns FAILURE, the sequence is aborted.
Super Type	ControlNode

SequenceStar	
Description	Representation of a sequence star control node. Use this control node when you don't want to tick children again that already returned SUCCESS.
Super Type	ControlNode

Parallel	
Description	Representation of a parallel control node. It allows concurrent execution of multiple branches.
Super Type	ControlNode

Fallback	
Description	Representation of a fallback control node. Fallback is a node that can express, as the name suggests, fallback strategies, i.e. what to do next if a child returns FAILURE.
Super Type	ControlNode

FallbackStar	
Description	Representation of a fallback start control node. FallbackStar will return RUNNING and the next time it is ticked, it will tick the same child where it left off before.
Super Type	ControlNode

Switch	
Description	Representation of a switch control node. Switch node represents the execution alternatives based on the value if the variable requested previously in Adaptation Engine. Depending of value of the variable, will be executed a child or other child.
Super Type	ControlNode

LeafNode	
Description	Abstract representation of the different leaf nodes. Available subclasses are as follows: ***AbstractAction*** ***Condition***
Super Type	TreeNode

AbstractAction	
Description	Abstract representation of the different actions. Available subclasses are as follows: ***AlwaysFailure*** ***AlwaysSuccess*** ***SetBlackboard*** ***Action***
Super Type	LeafNode

Condition	
Description	Condition node are equivalent to ActionNodes, but they are always atomic and synchronous, i.e. they must not return RUNNING. They should not alter the state of the system.
Super Type	LeafNode

AlwaysFailure	
Description	AlwaysFailure node always will return FAILURE.
Super Type	AbstractAction

AlwaysSuccess	
Description	AlwaysSuccess node always will return SUCCESS.
Super Type	AbstractAction

SetBlackboard	
Description	Representation of a set blackboard action. This node will store into blackboard the value of the variable stored in the input attribute.
Super Type	AbstractAction

Action	
Description	ActionNodes are leaves and do not have any children. The user should implement their own ActionNodes to perform the actual tasks.
Super Type	AbstractAction

DecoratorNode	
Description	Abstract representation of the different decorator nodes. Available subclasses are as follows: ***ForceSuccess*** ***Timeout*** ***Repeat*** ***Inverter*** ***ForceFailure*** ***RetryUntilSuccess*** ***BlackboardCheckDouble*** ***BlackboardCheckString*** ***BlakcboardCheckInt***
Super Type	TreeNode
Reference	***node***: contains the reference to the node being decorated.

ForceSuccess	
Description	If the child returns RUNNING, this node returns RUNNING too. Otherwise, it returns always SUCCESS.
Super Type	DecoratorNode

Timeout	
Description	All child of this node, will be terminated after the specified time.
Super Type	DecoratorNode

Repeat	
Description	Tick the child up to N times, where N is passed as a Input Port, as long as the child returns SUCCESS. Interrupt the loop if the child returns FAILURE and, in that case, return FAILURE too. If the child returns RUNNING, this node returns RUNNING too.
Super Type	DecoratorNode

Inverter	
Description	Tick the child once and return SUCCESS if the child failed or FAILURE if the child succeeded. If the child returns RUNNING, this node returns RUNNING too.
Super Type	DecoratorNode

ForceFailure	
Description	If the child returns RUNNING, this node returns RUNNING too. Otherwise, it returns always FAILURE.
Super Type	DecoratorNode

RetryUntilSuccess	
Description	Tick the child up to N times, where N is passed as a Input Port, as long as the child returns FAILURE. Interrupt the loop if the child returns SUCCESS and, in that case, return SUCCESS too. If the child returns RUNNING, this node returns RUNNING too.
Super Type	DecoratorNode

BlackboardCheckDouble	
Description	Representation of a blackboard check double decorator.
Super Type	DecoratorNode

BlackboardCheckString	
Description	Representation of a blackboard check string decorator.
Super Type	DecoratorNode

BlackboardCheckInt	
Description	Representation of a blackboard check int decorator.
Super Type	DecoratorNode

2.2 The MIRoN modelling language

The MIRoN Framework also includes an EMF-based MIRoN metamodel enabling the specification of:

1) the inputs that will feed the MIRoN Engine, namely the (internal and external) contextual information and the QoS metrics defined on relevant non-functional properties, both provided by the RoQME QoS Metrics Provider Component;
2) the outputs to be computed by the MIRoN Engine, namely the variation points to be fixed by the robot at runtime; and
3) the adaptation logic that explicate how the variation points (outputs) must be calculated in terms of the contextual information and QoS metrics provided by RoQME (inputs).

The MIRoN metamodel allows defining two different kinds of variation points, namely: (1) parameters (e.g., the robot maximum speed); and (2) alternative/optional behaviours.

- On the one hand, parameters are computed by the MIRoN Engine whenever the perceived situation (based on the contextual information and on the QoS metrics provided by RoQME), indicates the need to adjust them. Then, the affected parameters are recalculated and sent to the SmartSoft Skill Interface for the Sequencer to apply them.
- On the other hand, the alternative and optional behaviours described in the MIRoN models, are used to generate an extended BT model including the variants/options to be considered at runtime when reaching a node containing a variation point. When this happens, the BT interpreter selects the behaviour (subtree) recommended by the MIRoN Engine according to the perceived situation.

Based on the MIRoN metamodel, we have developed a textual model editor using Xtext allowing designers:

1) to import the contexts and QoS metrics, previously defined in a RoQME model;
2) to import a base BT model, describing the nominal behaviour of the robot (without adaptation);
3) to import the BT models describing variant/optional behaviours; and
4) to combine all the previous elements for defining a MIRoN model in terms of (i) contexts, based on the elements imported in 1; (ii) variation points (*varpoint*), either alternative/optional behaviours based on the elements imported in 2 and 3, or parameters; and (iii) adaptation rules (*rule*) and quality impacts relating context and variation points so that the former determine how the later are reconfigured.

Figure 3 shows an example of the developed textual editor. Moreover, Figure 4 and *Figure 5* presents the MIRoN metamodel. It is worth noting that the MIRoN metamodel is an extension of the RoQME metamodel and, as a consequence, all the RoQME modelling concepts are available in MIRoN. The MIRoN metamodel is divided into two parts, namely, Data Types and Kernel. Next, the concepts included in each of these parts are described.

a) The Data Types metamodel

RoQMEModel (from the RoQME metamodel)	
Description	Root element of a RoQME model
Super Type	DocumentableElement
Attributes	***namespace***: univocal name to identify the RoQME model
References	***datatypes***: it contains a set of data type definitions that can be used when declaring variables
	variables: it contains a set of variable declarations
	sentences: it contains a set of sentences (e.g. observations)

intralogistics.miron

```
import "intralogistics.bt" as main
import "intralogistics.roqme" as roqme

varpoint task : enum {
    pick : main.PickDeliverCharge,
    deliver : main.DeliverCharge,
    dock : main.Charge,
    pickDeliverCharge : preset
} in main.PickDeliverChargeSubtree

varpoint maxVelocity : number (10:1:100) {
    10 affects roqme.safety++,
    100 affects roqme.mission_completion++
} in arch {
    parameterSetRepository "CommNavigationObjects"
    parameterSet "CdlParameter"
    component "SmartCdlServer"
    parameter "transvel"
}

rule noBattery : roqme.power_autonomy < 0.3
    implies task = task::dock
```

Figure 3. Example of the MIRoN model editor

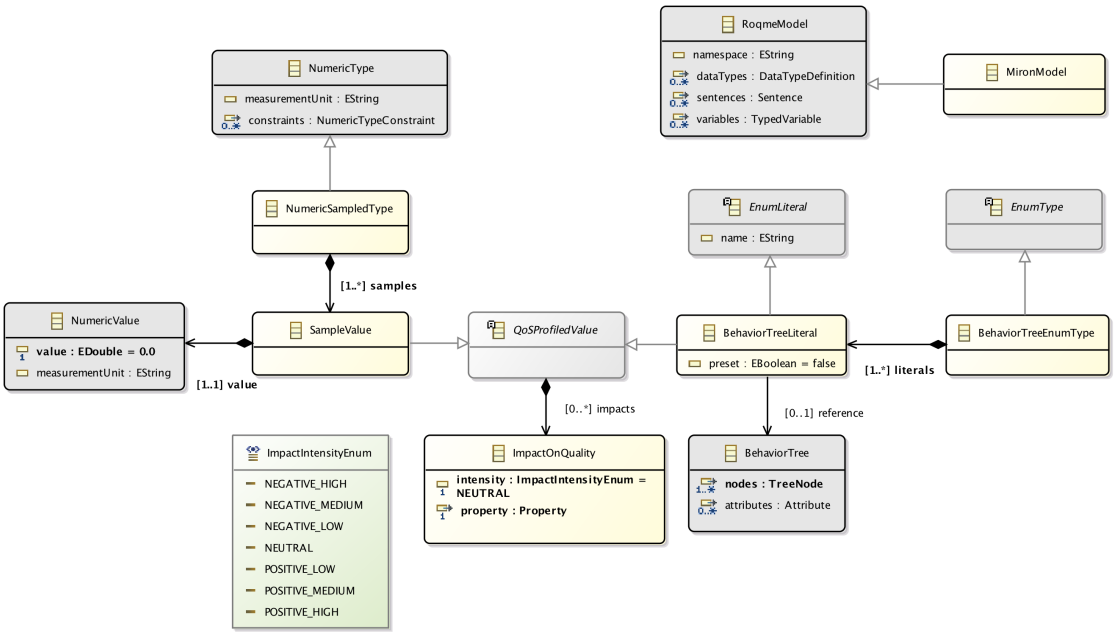

Figure 4. Data Types part of the MIRoN Metamodel

MironModel	
Description	Root element of a MIRoN model.
Super Type	RoqmeModel

NumericType (from the RoQME metamodel)	
Description	It represents a numeric data type.
Super Type	DataType

NumericSampledType	
Description	A numeric type that can declare the quality impact on non-functional properties that some sample values have.
Super Type	NumericType
References	***samples***: one or more quality impacts declaration.

QoSProfiledValue	
Description	Abstract class to declare a quality impact on a non-functional property.
References	***impacts***: quality impacts

ImpactOnQuality	
Description	Defines a quality impact on a non-functional property.
References	***intensity***: the impact influence ***property***: non-functional property on which the impact acts

SampleValue	
Description	A numeric value with a quality impact definition.
Super Type	QoSProfiledValue
References	***value***: a numeric value

NumericValue (from the RoQME metamodel)	
Description	Represents a concrete number.
Super Type	SingleValue
References	***value***: a real number. ***measurementUnit***: optional text to indicate the unit of measurement

EnumType (from the RoQME metamodel)	
Description	Identifies an enumerated data type.
Super Type	DataType

BehaviorTreeEnumType	
Description	Represents an enumerated type whose elements point to BehaviorTree instances (see Behavior Tree metamodel) and declare quality impacts on non-functional properties.
Super Type	EnumType
References	***literals***: contains the set of elements that defines this enumerated type

EnumLiteral (from the RoQME metamodel)	
Description	It models an enumerated literal.
Attributes	***name***: literal identifier

BehaviorTreeLiteral	
Description	Represents an enumerated literal, associated with a BehaviorTree instance, that defines a quality impact on a non-functional property.
Super Type	EnumLiteral
Attributes	***preset***: marks this element as the default option
References	***reference***: BehaviorTree instance.

BehaviorTree (from the BahaviorTree metamodel)	
Description	Representation of a BT (see Section 2.1).

b) The Kernel metamodel

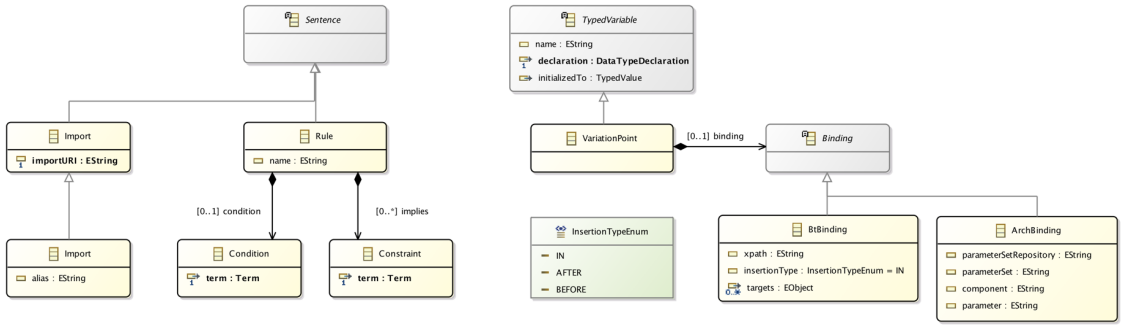

Figure 5. Kernel part of the MIRoN Metamodel

Sentence (from the RoQME metamodel)	
Description	Abstract class to represent sentences in the language
Super Type	DocumentableElement

Import (form the RoQME metamodel)	
Description	Allows importing other models to reuse definitions.
Super Type	Sentence
Attributes	***ImportURI:*** URI of the model to be imported.

Import	
Description	Allows importing other models with an alias that can be used as a custom namespace.
Super Type	Import
Attributes	***alias:*** custom namespace identifier.

Rule	
Description	Represents an adaptation rule.
Super Type	Sentence
Attributes	***name***: name of the rule.
References	***condition:*** contextual condition to trigger the rule. ***implies***: constraints to be applied when the rule is triggered.

Condition	
Description	Expression that establishes the trigger condition of a rule.
References	***term***: math expression (defined in terms of the RoQME metamodel)

Constraint	
Description	Expression that establishes the consequences of applying a rule.
References	***term***: math expression (defined in terms of the RoQME metamodel)

TypedVariable (from the RoQME metamodel)	
Description	It abstracts a variable belonging to a data type.
Super Type	DocumentableElement
Attributes	***name***: identifies the variable.
References	***declaration***: contains the kind of data type declaration for the variable ***initializedTo***: value to which the variable is initialized

VariationPoint	
Description	Defines a variation point variable, i.e., a variable that can be modified as a result of the adaptation process.
Super Type	TypedVariable
References	***binding:*** link to the system element that receives the changes.

Binding	
Description	Abstract class to identify a binding point in the system to which a variation point applies.

BtBinding	
Description	Links a variation point to one or more nodes in a behaviour tree.
Super Type	Binding
Attributes	***xpath***: XPath query to select the nodes that will be affected by the variation point. ***insertionType***: indicates the position where a variant can be inserted in the behaviour tree.
References	***targets***: node references. It can be filled manually by the user or automatically through the execution of a XPath query.

ArchBinding	
Description	Links a variation point to a component parameter in the system architecture.
Super Type	Binding
Attributes	***parameterSetRepository:*** repository identifier. ***parameterSet:*** parameter set identifier. ***component:*** component identifier. ***parameter:*** parameter identifier.

2.3 Model generators

The model generators included in the MIRoN Framework are the following:

- An extended BT model generator has been implemented to weave the alternative/optional behaviours within the nominal BT model, as indicated in the input MIRoN Model.
- A MiniZinc [8] model generator has also been implemented to encode the adaptation logic described in an input MIRoN Model, so that it can be readily used at runtime by the MiniZinc Interpreter when invoked by the MIRoN Adaptation Engine.

3 The MIRoN Adaptation Engine

The MIRoN Adaptation Engine has been developed as a C++ program. It is subscribed (via DDS) to the contexts and QoS metrics monitored by the RoQME QoS Metrics Provider component. When a change is identified in any of these inputs, it sets the corresponding MiniZinc parameter with the new value and runs the MiniZinc Interpreter [8]. This interpreter recalculates the affected variation points and, in case a reconfiguration is needed, it sends (via the ZMQ Broker) the recommended adaptation either to the BT Executor (in case the variation point implies a change in the course of action), or to the SmarSoft Skill Interface (in case the variation point is bound to a parameter). The ZeroMQ-based Broker enables the communication among the MIRoN Engine, the BT executor and the SmartSoft Skill Interface. Figure 6 summarizes the different elements involved in the operation of the Adaptation Engine.

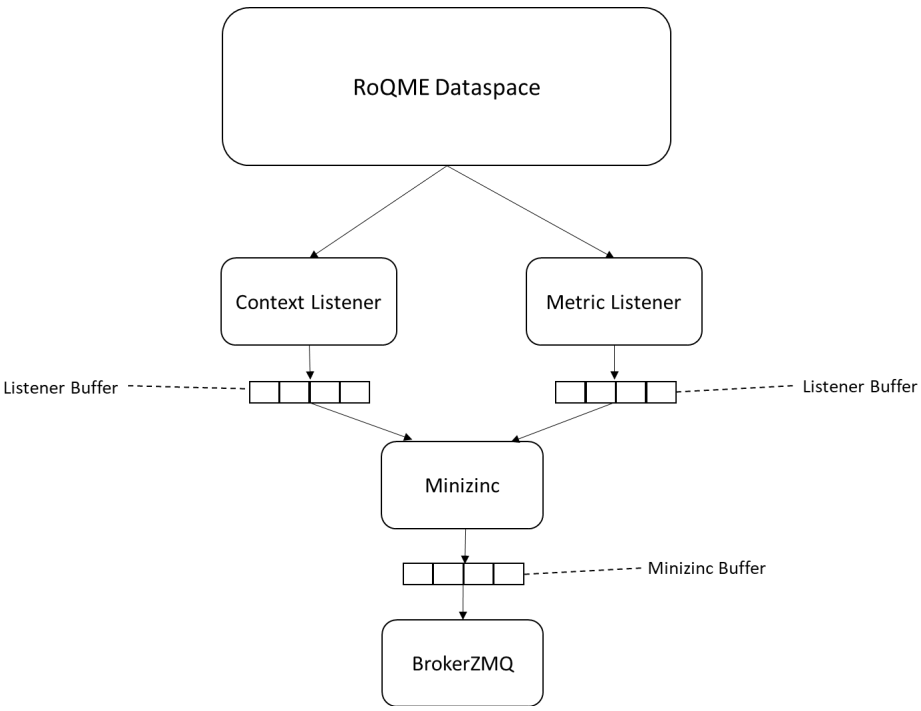

Figure 6. Operation of the Adaptation Engine

References

[1] "The MIRoN ITP", 2020. [Online]. Available: https://robmosys.eu/MIRoN.

[2] "The MOOD2BE ITP", 2020. [Online]. Available: https://robmosys.eu/mood2be.

[3] "The MOOD2BE Tools", 2020. [Online]. Available: https://github.com/BehaviourTree.

[4] "The RoQME ITP", 2020. [Online]. Available: https://robmosys.eu/roqme/.

[5] "The RoQME Toolchain", 2019. [Online]. Available: https://github.com/roqme.

[6] "The SmartSoft MDSD Toolchain", 2020 [Online]. Available: https://wiki.servicerobotik-ulm.de/smartmdsd-toolchain:start

[7] Xtext, 2020 [Online]. Available: https://www.eclipse.org/Xtext/

[8] Minizinc, 2020 [Online]. Available: https://www.minizinc.org/