

**Design Document for the
hyades-models System**

Table Of Contents

1. SCOPE	4
1.1 PURPOSE	4
1.2 SYSTEM OVERVIEW.....	4
1.3 DOCUMENT OVERVIEW	4
2. REFERENCED DOCUMENTS	4
3. ARCHITECTURAL GOALS AND CONSTRAINTS	4
4. LOGICAL ARCHITECTURE	4
4.1 OVERVIEW.....	4
4.2 HIERARCHY	5
4.2.1 Overview.....	5
4.2.2 Classes.....	11
4.3 EXTENSIONS	13
4.3.1 Overview.....	13
4.3.2 Classes.....	15
4.4 SDB.....	16
4.4.1 Overview.....	16
4.4.2 Classes.....	17
4.5 CBE.....	18
4.5.1 Overview.....	18
4.5.2 Classes.....	20
4.6 COMMON	21
4.6.1 Overview.....	21
4.6.2 Classes.....	23
4.7 BEHAVIOR.....	24
4.7.1 Overview.....	24
4.7.2 Classes.....	24
4.8 INTERACTIONS	24
4.8.1 Overview.....	24
4.8.2 Classes.....	25
4.9 FRAGMENTS.....	26
4.9.1 Overview.....	26
4.9.2 Classes.....	29
4.10 TESTPROFILE.....	30
4.10.1 Overview.....	30
4.10.2 Classes.....	34
4.11 CONFIGURATION	37
4.11.1 Overview.....	37
4.11.2 Classes.....	40
4.12 DATAPOOL	43
4.12.1 Overview.....	43
4.12.2 Classes.....	45
4.13 TRACE.....	45
4.13.1 Overview.....	45
4.13.2 Classes.....	53
4.14 STATISTICAL	56
4.14.1 Overview.....	56
4.14.2 Classes.....	57
4.15 TEST	59

4.15.1	Overview.....	59
4.15.2	Classes.....	59
4.16	DATA.....	59
4.16.1	Overview.....	59
4.16.2	Classes.....	60
4.17	ECORE.....	61
4.17.1	Overview.....	61
4.17.2	Classes.....	66
5.	INTERACTION DIAGRAMS.....	68
5.1	TOP LEVEL INTERACTION DIAGRAMS.....	68
5.2	HIERARCHY.....	68
5.3	SDB.....	68
5.4	CBE.....	68
5.5	COMMON.....	68
5.6	TRACE.....	68
5.7	STATISTICAL.....	68
5.8	TEST.....	68
5.9	ECORE.....	68

1. SCOPE

1.1 Purpose

1.2 System Overview

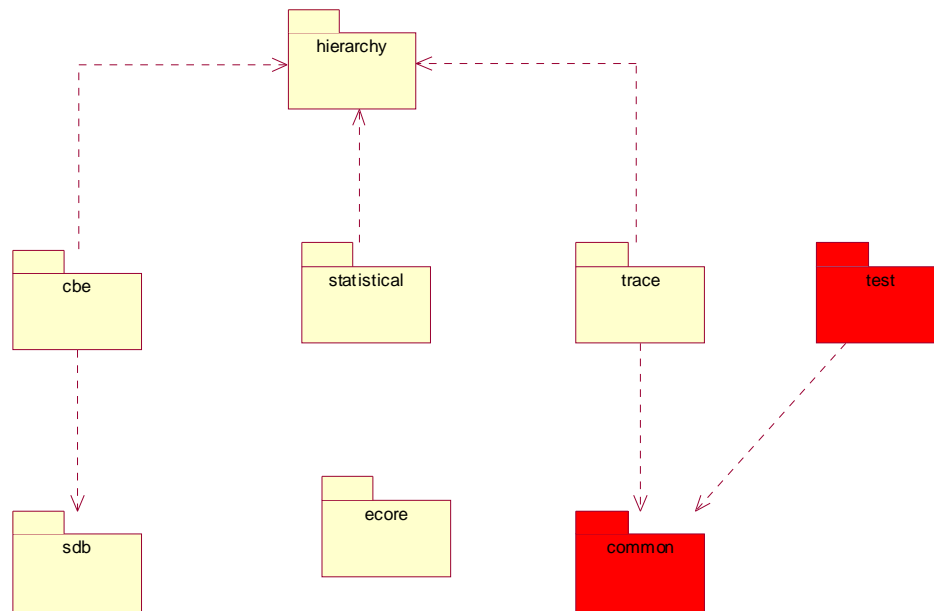
1.3 Document Overview

2. REFERENCED DOCUMENTS

3. ARCHITECTURAL GOALS AND CONSTRAINTS

4. LOGICAL ARCHITECTURE

4.1 Overview



Conventions:

- Classes and associations with a fill color of RED are still under design.
- Classes with a fill color of grey are from another package. Please see that package for details.
- Classes with a fill color of green have a very close mapping to an external standard
- Classes with no fill color are from the same package, but are primarily defined in another class diagram. When editing that Class, please use the primary diagram where the Class will have the default fill color.
- All associations are assumed to be ordered unless noted otherwise in the documentation for the association.
- These packages are intended to be the source for EMF based code generation.
- All classes except roots will be owned by value in at least one place in the model. If owned by value in multiple places, these associations are mutually exclusive. This simplifies mapping persisted location to the type defined in this model, as well as reduces coding by the model users.
- GUIDs as used in this model are not specific to any OS implementation. It is up to the data provider to provide globally unique identifiers as it sees fit.

Time:

Most representations of time in this model are of type "double." Unless otherwise noted in the documentation for an attribute, time is measured as a number of microseconds since the start of January 1, 1970 UTC. (It's a double, so the measurement can be as precise as the underlying system allows.)

Some "time" attributes are deltas from other times: in these cases the attribute's documentation tells what other instance and attribute form the base time that the delta is measured from.

A convention is required to define how optional data, or uncollected data is indicated in the model structure and instances of the model.
 Although language neutrality is the end goal for this model, it needs to be visited for every part of at least the trace model for each language we introduce.

Figure 1: Main

4.2 hierarchy

4.2.1 Overview

Internal package, please do not extend or use.

The "hierarchy" package is used to capture the base topology used to provide a hierarchical view that is used to organize and interact with the various agent contents.

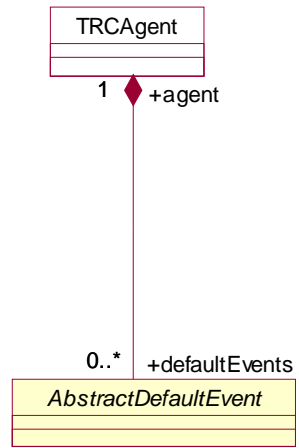


Figure 2: CBE relations

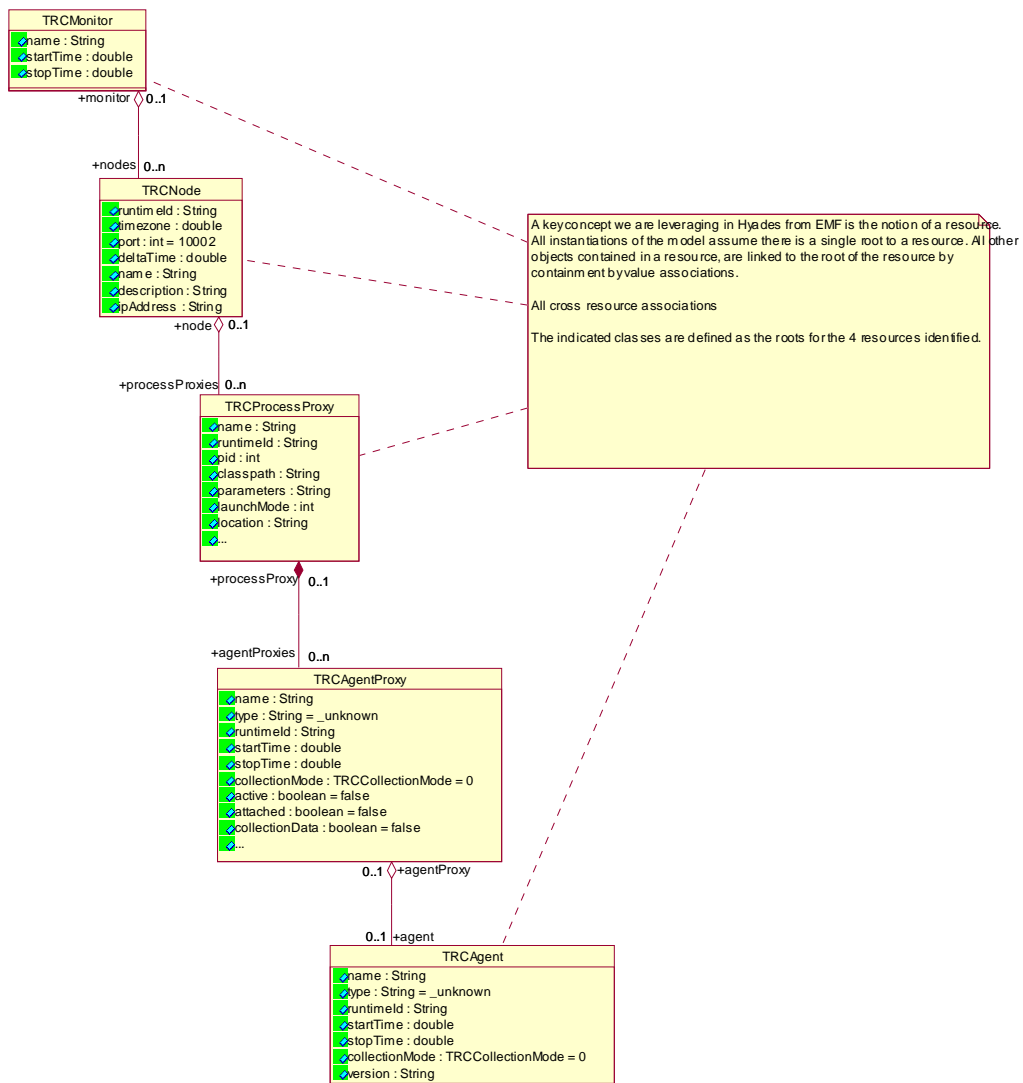


Figure 3: Monitor

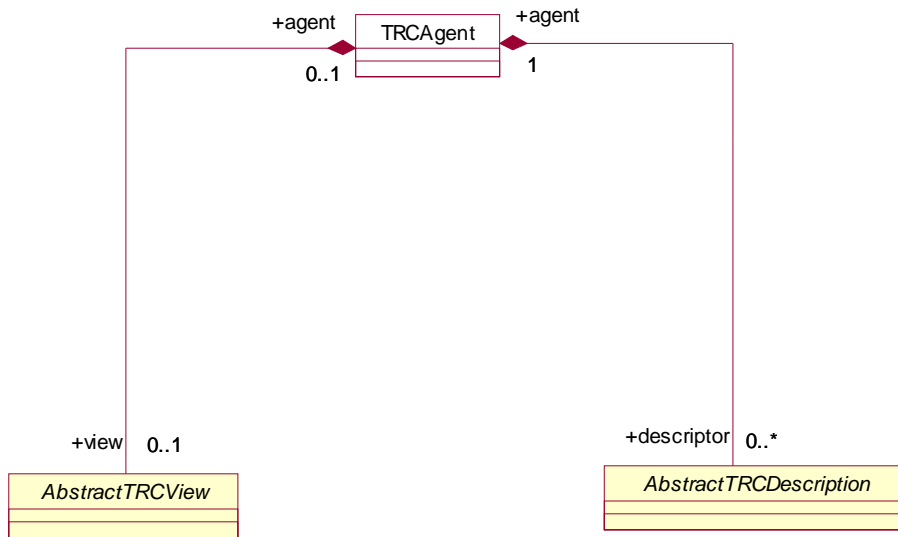


Figure 4: Statistical relations

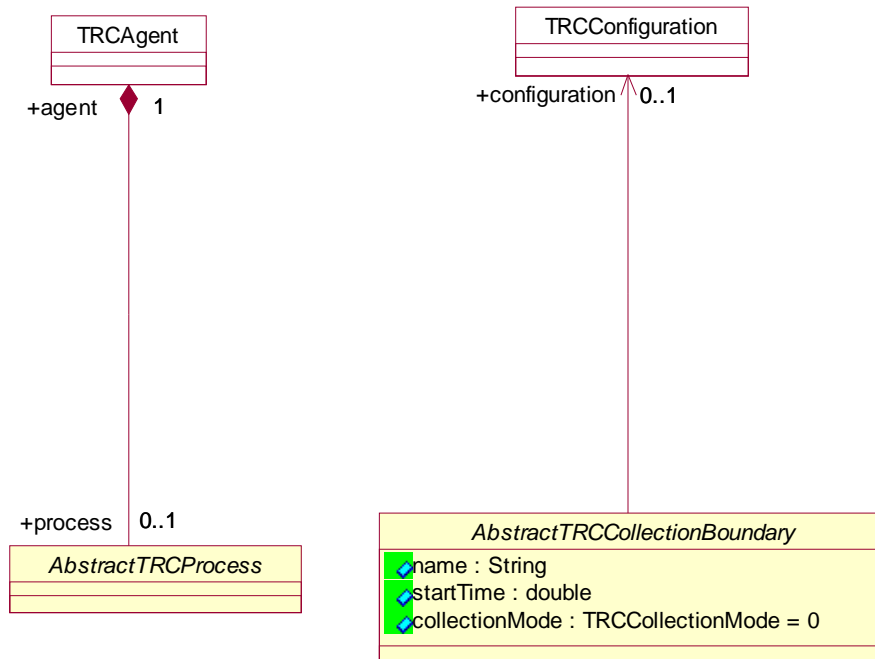


Figure 5: Trace relations

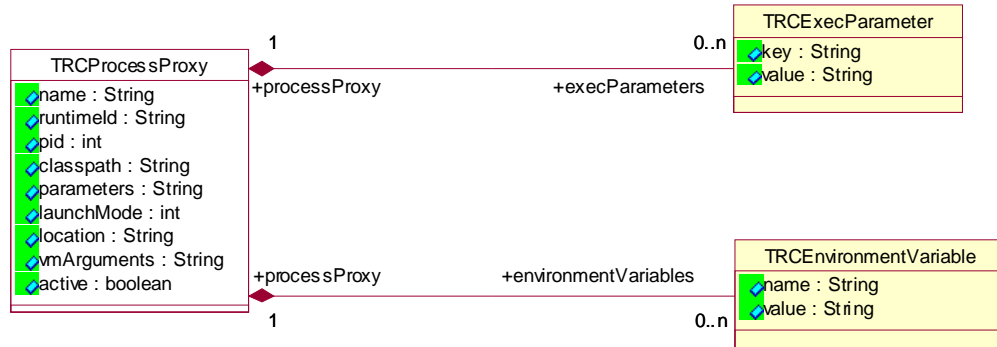


Figure 6: Process Proxy Extensions

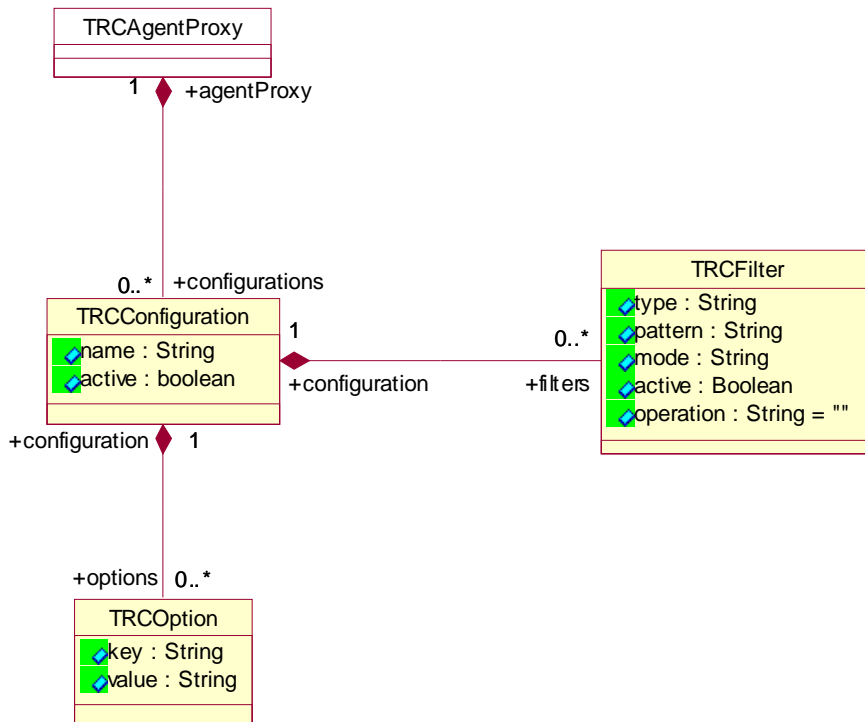


Figure 7: Agent Proxy Extensions

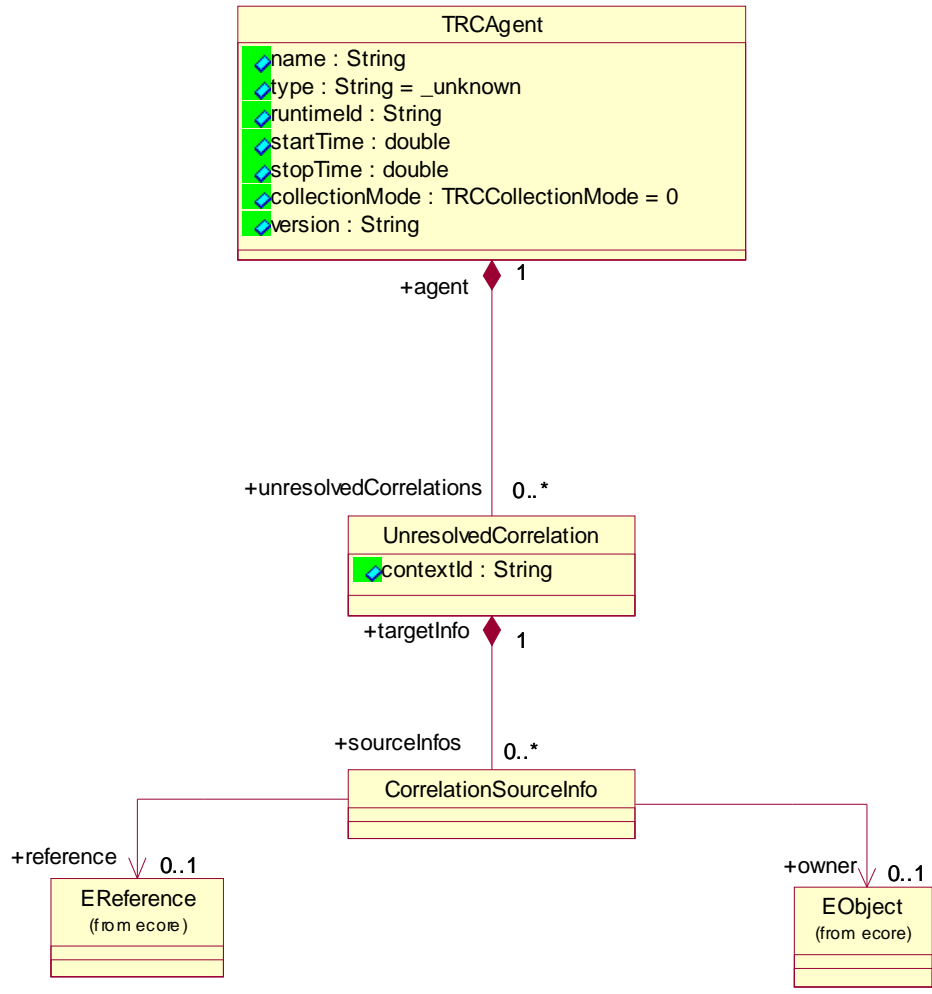


Figure 8: Unresolved references

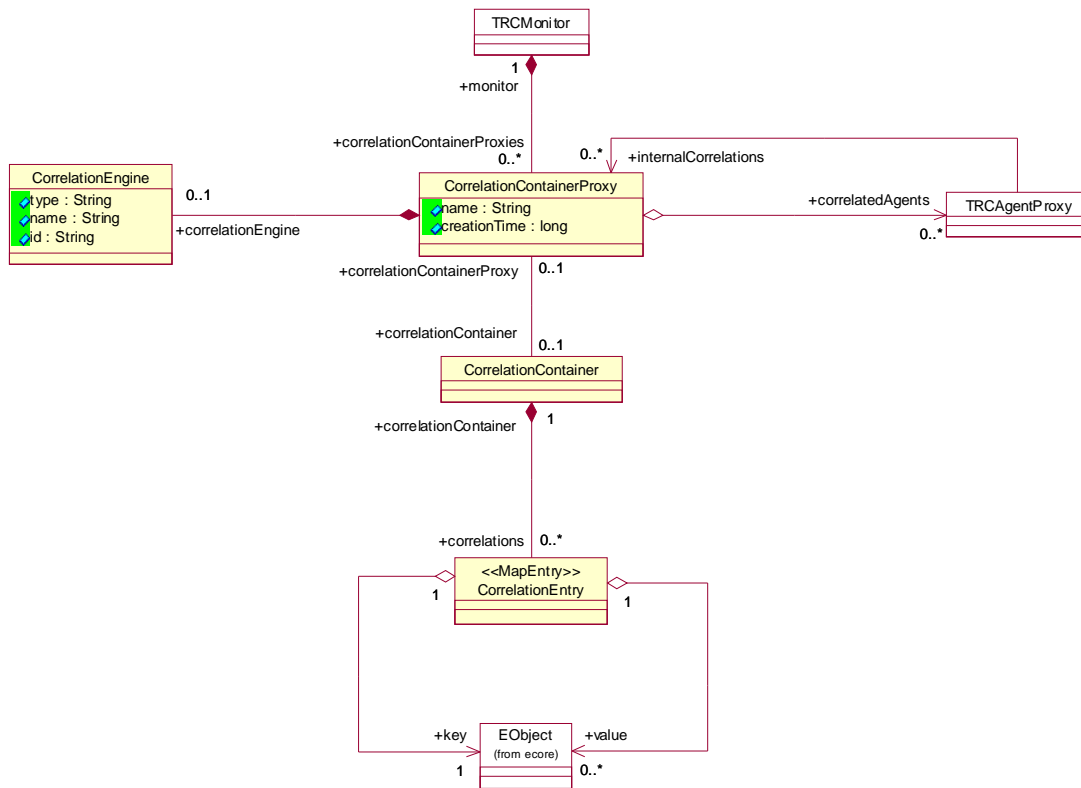


Figure 9: Correlations

4.2.2 Classes

4.2.2.1 TRCCollectionMode (NormalClass)

4.2.2.2 TRCProcessProxy (NormalClass)

The proxy for a real OS process. Has properties that represent the execution context. The reason this is referred to as a proxy is that if an agent is actually collecting an execution trace, it will contain the relevant details at runtime of the process that is being monitored.

4.2.2.3 TRCOption (NormalClass)

This is a name/value pair, used to define options for the profiling agent. Eg.:
key="ALLOCATION_INFORMATION" optionValue="all"

4.2.2.4 TRCAgent (NormalClass)

An agent is analogous to a "trace file". An agent is typed to hold a particular type of trace, and the TRCAgent object owns by value the instances of the trace data. Because agents are typed, there can be zero or more agents associated with a given TRCMonitor, TRCNode or TRCProcessProxy. However, a given agent instance can only be associated with one of those objects.

This agent is the model entity for a data provider. Providers provide metadata as a collection of descriptors and the data itself as observations.

The agent is also the granularity of persistence.

4.2.2.5 TRCAgentProxy (NormalClass)

A TRCAgent proxy, to enable support for loading and unloading the agent resource. This class shadows most of the properties of the TRCAgent: please check the documentation on TRCAgent for duplicated fields.

4.2.2.6 TRCConfiguration (NormalClass)

A configuration is used to hold a set of options and filters. This would normally be used to capture a set of options and filters that are active for a given agent.

4.2.2.7 TRCEnvironmentVariable (NormalClass)

A name-value pair to capture an environment variable. For example classpath

4.2.2.8 TRCExecParameter (NormalClass)

4.2.2.9 TRCFilter (NormalClass)

A filter describes a pattern that can be used during data collection to control the data being collected.

4.2.2.10 TRCNode (NormalClass)

A Node is a machine, or at least a machine execution partition. It owns process proxies for processes with agents attached and data collected.

There is one server/service installed for each Node. An analogy is a service running on NT that hooks to the process being monitored.

4.2.2.11 TRCMonitor (NormalClass)

A monitor is in effect a complete set of traces that may have been collected from one or more agents. It is the logical root for a persisted model instance that may reflect a complete distributed application or simply a set of data collection points.

4.2.2.12 AbstractDefaultEvent (NormalClass)

This class allows for a clean plugin dependency packaging. Hyades needs to allow different plugins for each model sub-package in order to avoid all of them having to always be present and possibly loaded.

4.2.2.13 AbstractTRCView (NormalClass)

This class allows for a clean plugin dependency packaging. Hyades needs to allow different plugins for each model sub-package in order to avoid all of them having to always be present and possibly loaded.

4.2.2.14 AbstractTRCDescription (NormalClass)

This class allows for a clean plugin dependency packaging. Hyades needs to allow different plugins for each model sub-package in order to avoid all of them having to always be present and possibly loaded.

4.2.2.15 AbstractTRCProcess (NormalClass)

This class allows for a clean plugin dependency packaging. Hyades needs to allow different plugins for each model sub-package in order to avoid all of them having to always be present and possibly loaded.

4.2.2.16 AbstractTRCCollectionBoundary (NormalClass)

This class allows for a clean plugin dependency packaging. Hyades needs to allow different plugins for each model sub-package in order to avoid all of them having to always be present and possibly loaded.

4.2.2.17 UnresolvedCorrelation (NormalClass)

Class used to persist the unresolved correlations. When a correlation is resolved the object should be removed from the list.

4.2.2.18 CorrelationSourceInfo (NormalClass)

Describes the known information from the received event.

4.2.2.19 CorrelationContainer (NormalClass)

This is a root in a correlation instance resource. For example the correlation instance is created when a correlation operation/action is used or by the loaders when an AssociationEngine event is received.

4.2.2.20 CorrelationContainerProxy (NormalClass)

A UI proxy class, to avoid loading the whole correlation instance resource.

4.2.2.21 CorrelationEntry (NormalClass)

Represents a 1-many relation between a source event and it's associated events (an event can be a CBE, method invocation etc)

4.2.2.22 CorrelationEngine (NormalClass)

The id should uniquely identify the engine.

4.2.2.23 BooleanArray (NormalClass)

4.2.2.24 IntArray (NormalClass)

4.3 extensions

4.3.1 Overview

Resource extensions

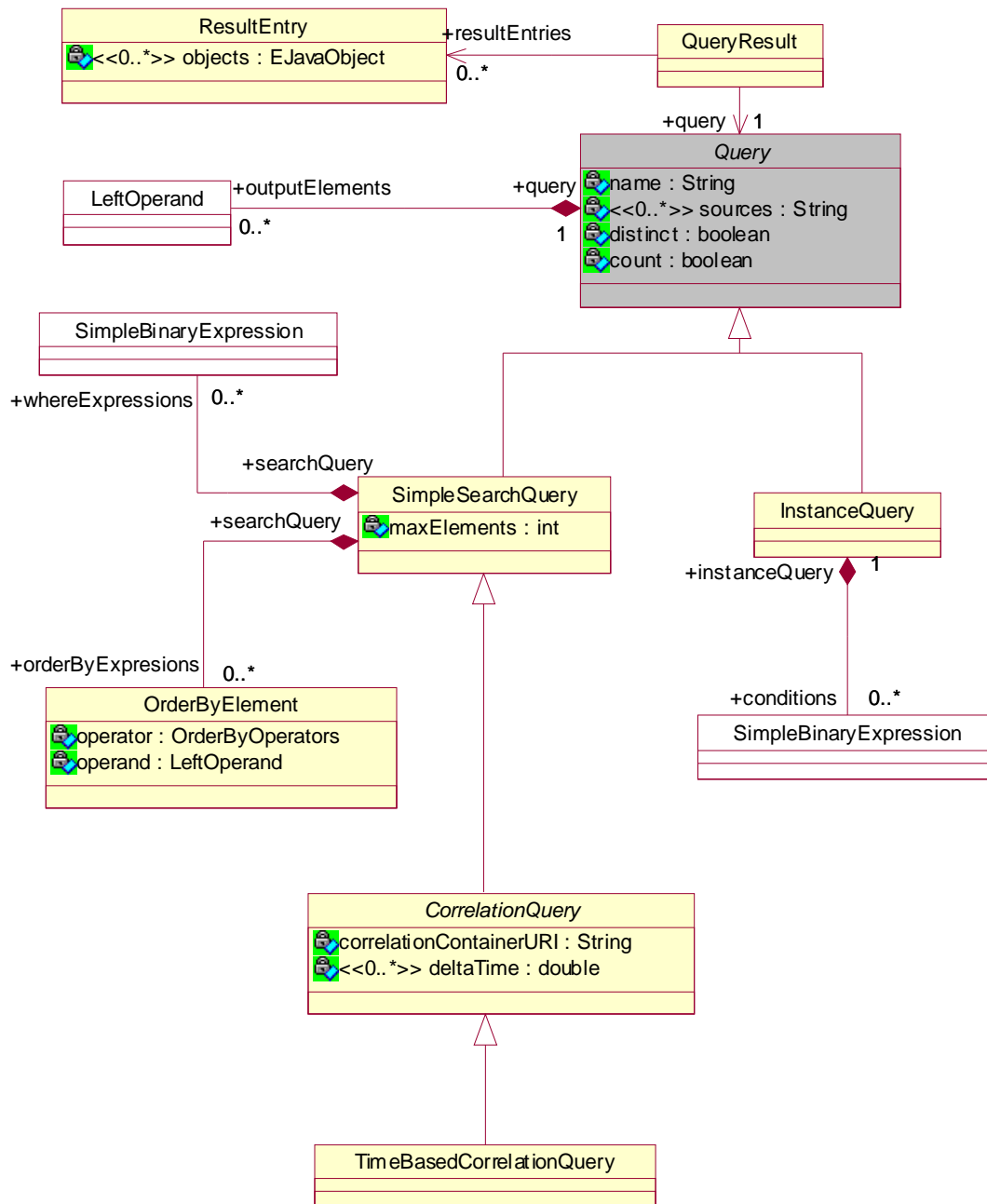


Figure 10: Query support

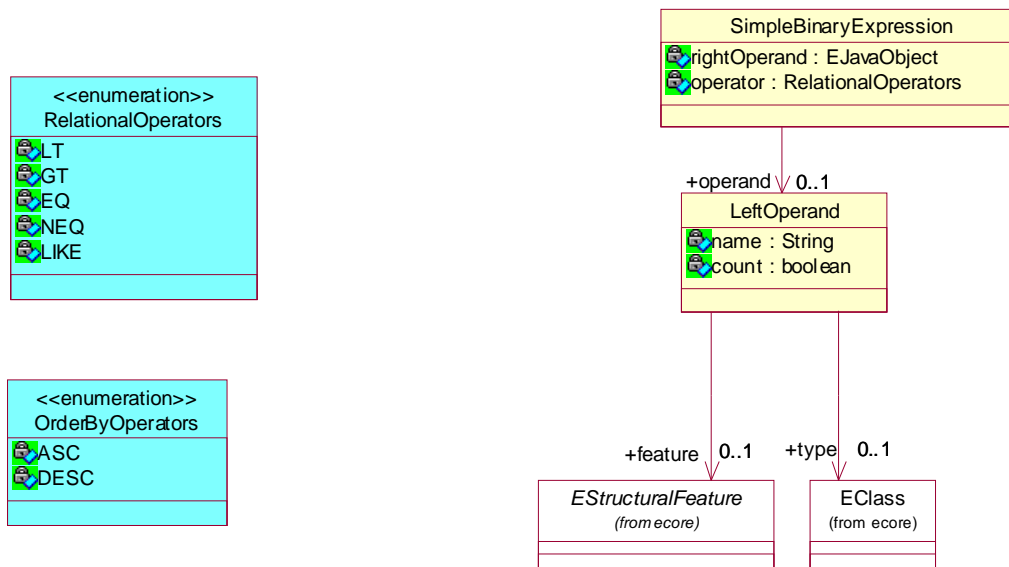


Figure 11: Expressions, Operands and Operators

4.3.2 Classes

4.3.2.1 Query (NormalClass)

4.3.2.2 SimpleSearchQuery (NormalClass)

4.3.2.3 InstanceQuery (NormalClass)

A simple query, in XMI case could have an optimized implementation

4.3.2.4 OrderByElement (NormalClass)

4.3.2.5 QueryResult (NormalClass)

Mixed list of output elements instances.

4.3.2.6 SimpleBinaryExpression (NormalClass)

4.3.2.7 LeftOperand (NormalClass)

4.3.2.8 RelationalOperators (NormalClass)

4.3.2.9 OrderByOperators (NormalClass)

4.3.2.10 ResultEntry (NormalClass)

4.3.2.11 CorrelationQuery (NormalClass)

4.3.2.12 TimeBasedCorrelationQuery (NormalClass)

4.4 *sdb*

4.4.1 Overview

Internal package, please do not extend or use.

The symptom data base is used to contain symptoms to be used by an analysis engine whne analyzing common base events

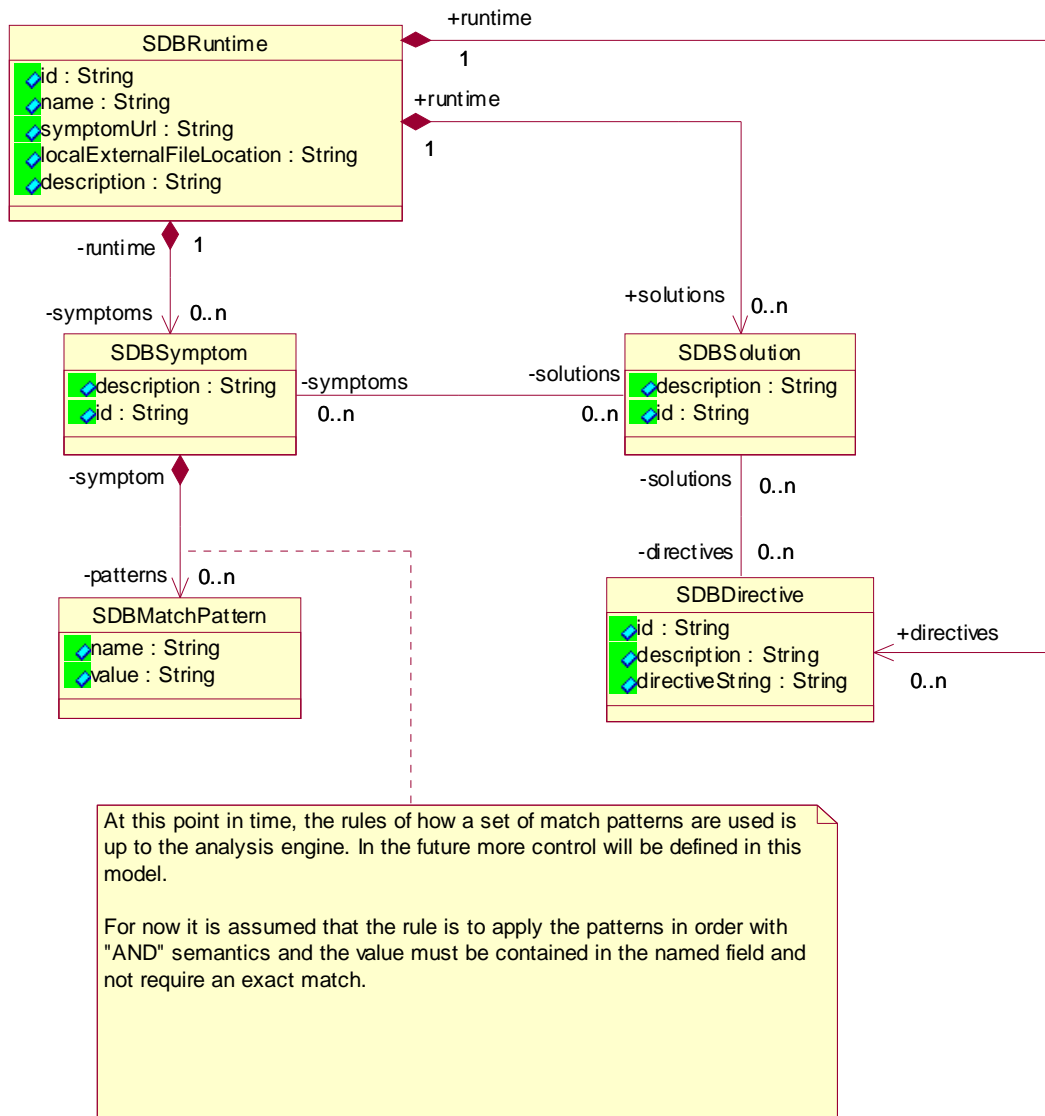


Figure 12: Symptom Database

4.4.2 Classes

4.4.2.1 SDBRuntime (NormalClass)

A runtime instance typically represents a product or a system, such as a Relational Database or a web server.

4.4.2.2 SDBSymptom (NormalClass)

A symptom is a unique set of match criteria that can be used to identify a problem. The uniqueness of the symptom is up to the instance provider.

4.4.2.3 SDBMatchPattern (NormalClass)

Match patterns are used in sequence to determine if a event being analyzed meets the criteria of a symptom.

4.4.2.4 SDBSolution (NormalClass)

A solution is a action that can be taken to address the symptom. A solution can take the form of a message to a user or automated actions, and that is all defined by the directives.

Solutions are owned by and unique to a runtime.

4.4.2.5 SDBDirective (NormalClass)

Directives are the actual steps or actions that are recommended for a given solution.

Although not required to be mutually exclusive the description and directive string are often not both used at the same time.

4.5 *cbe*

4.5.1 Overview

The Common Base Event model is used to describe a basic model for storing "messages" or logs from a product. The Common Base Event is the core structure for holding such messages which are often referred to as events from the products that log them.

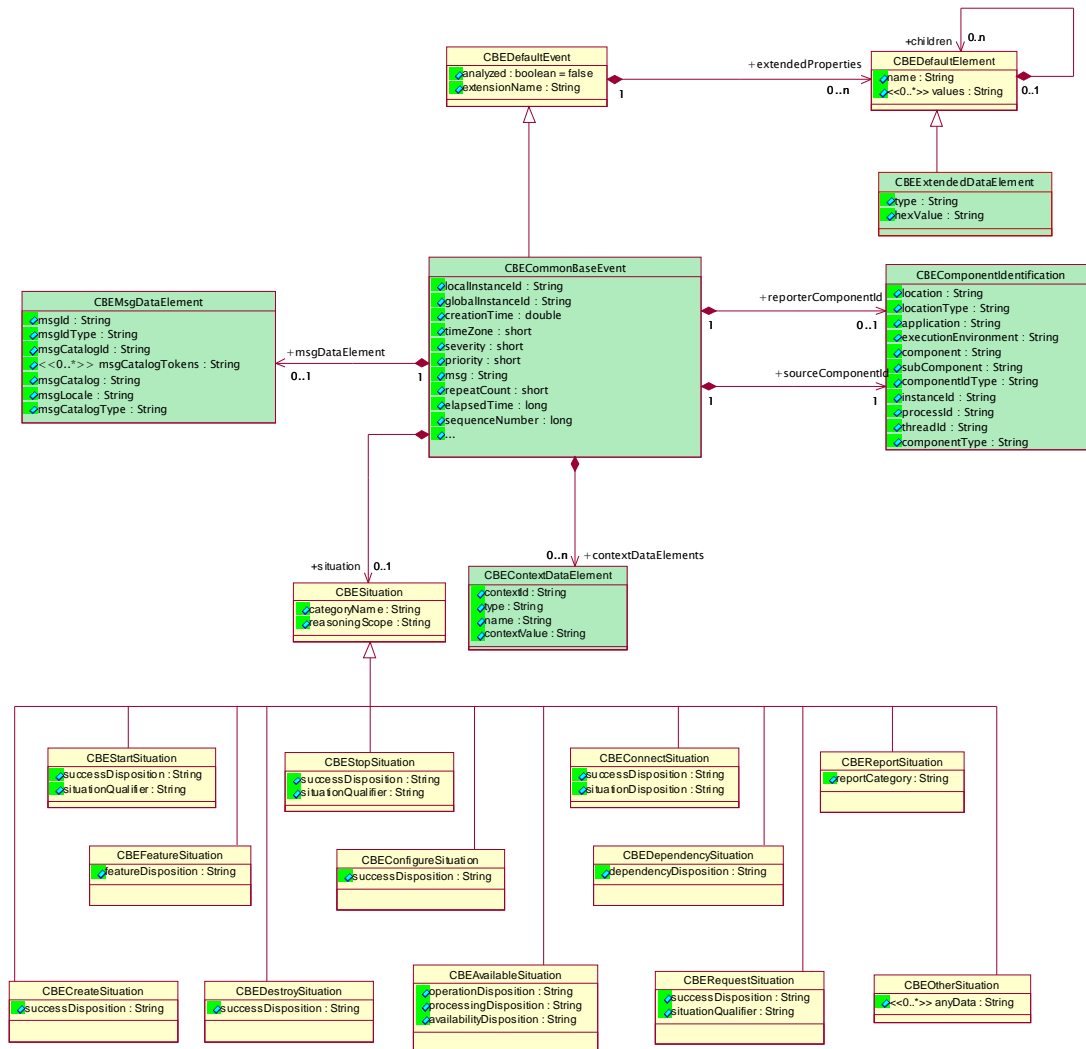


Figure 13: Main

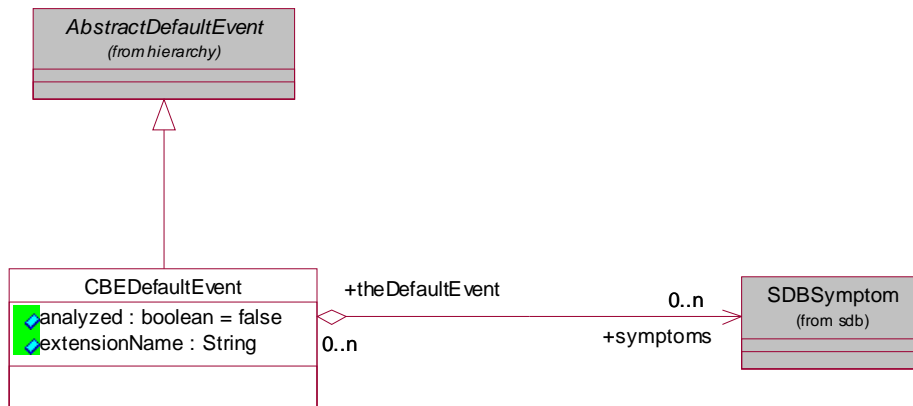


Figure 14: Analysis

4.5.2 Classes

4.5.2.1 CBECommonBaseEvent (NormalClass)

This class provides the the basic level of common properties for all problem artifacts.

4.5.2.2 CBEExtendedDataElement (NormalClass)

The ExtendedDataElement class provides a name-value pair for a Common Base Eevent. For example you may choose to have a name-value pair to provide return error code, extended problem situation description, or provide any additional data for the consumer of the event.

Token names may be referenced by correlation rules, however the ContextDataElement's prime purpose is ot hold data used for correlation. The support of this class relieves management tools of the need to be able to tokenize application specific artifacts. The token definitions are assumed to be specific to a resource. The data format is assumed to be in big endian.

4.5.2.3 CBEContextDataElement (NormalClass)

The Context Data Element provides data that is used ot establish the envrionment of an event. This data is intended to be used when post processing the model in order to associate or "correlate" multiple events together.

4.5.2.4 CBEComponentIdentification (NormalClass)

A component is a modular unit of a running system that has been defined to be a source of events, or a reporter of events. Typically these are large grained parts of a system, or services that are provided to a system. However the notion of a component is up to the system itslef to define.

4.5.2.5 CBEMsgDataElement (NormalClass)

When an event is used to represent a message, a message data element captures the several parts of the message information. For example "RDB2742E: an Error was detected in foo indicating that bar had problems" which has a msg id as well as substituted "foo" and "bar" variables.

4.5.2.6 CBEDefaultEvent (NormalClass)

A default event is the parent of any class that you wish to pass to an analysis engine. It will be marked isAnalyzed by the engine as appropriate. The default event is also the most minimal event that can be logged. As an instance, all values logged are held in a structure of default elements.

4.5.2.7 CBEDefaultElement (NormalClass)

This is the most primitive holder of data. A simple name value pair.

4.5.2.8 CBESituation (NormalClass)

4.5.2.9 CBEStartSituation (NormalClass)

4.5.2.10 CBEConnectSituation (NormalClass)

4.5.2.11 CBEStopSituation (NormalClass)

4.5.2.12 CBEReportSituation (NormalClass)

4.5.2.13 CBEFeatureSituation (NormalClass)

4.5.2.14 CBEConfigureSituation (NormalClass)

4.5.2.15 CBEDependencySituation (NormalClass)

4.5.2.16 CBECreateSituation (NormalClass)

4.5.2.17 CBEDestroySituation (NormalClass)

4.5.2.18 CBEAvailableSituation (NormalClass)

4.5.2.19 CBERequestSituation (NormalClass)

4.5.2.20 CBEOtherSituation (NormalClass)

4.6 *common*

4.6.1 Overview

Internal package, please do not extend or use.

The common package provides the base classes that are similar to the UML concepts for defining classes and behaviour. This package is not intended to provide a general meta model for these concepts. It will be replaced if a UML model were to appear in Eclipse that can be used.

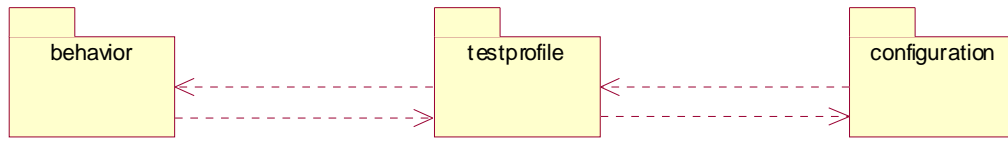


Figure 15: Package Overview

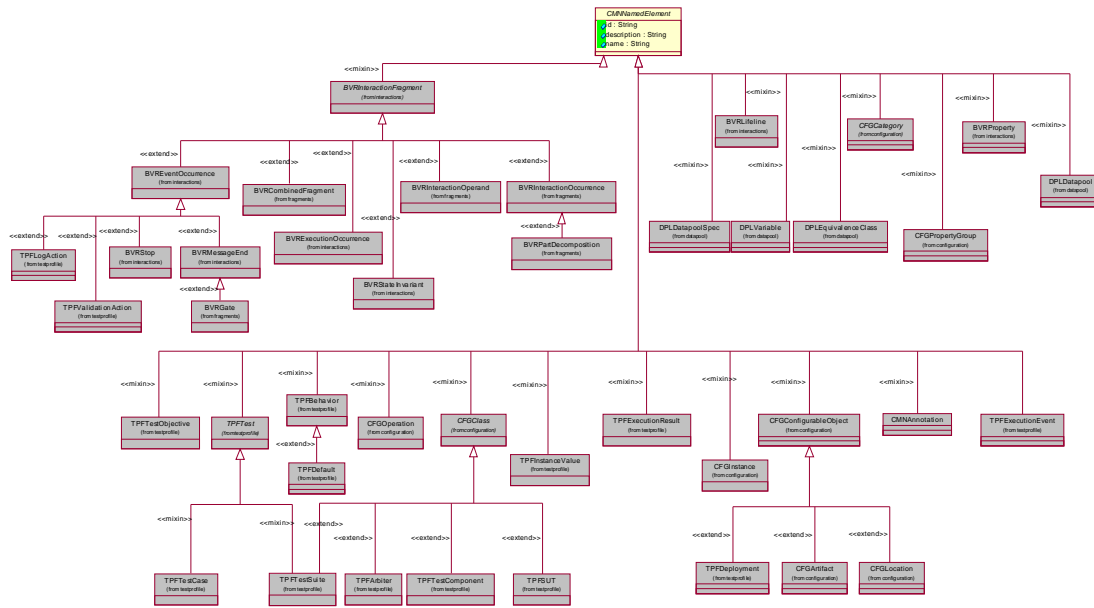


Figure 16: Hierarchy

CMNDefaultProperty is a property class with no methods and no attributes. Its role is as a tag interface within the model, to allow for future use of SDOs or Dynamic EMF classes to be used as properties.

Any Hyades participating vendor should discuss the use of this class with the Test Model team before utilizing it. This class should only be used to store properties that are not of general interest to the test model. Any properties that are of general interest should be discussed with the test model team, and upon agreement, added to the model directly.

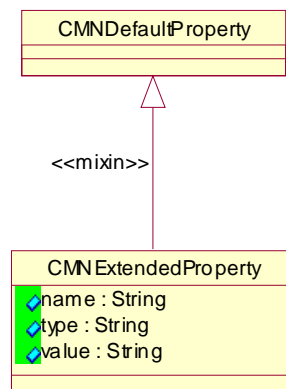


Figure 17: Properties

4.6.2 Classes

4.6.2.1 CMNNamedElement (NormalClass)

4.6.2.2 CMNMachine (NormalClass)

4.6.2.3 CMNNodeType (NormalClass)

4.6.2.4 CMNNodeInstance (NormalClass)

4.6.2.5 CMNExtendedProperty (NormalClass)

This class stores the name, type and value of an extended property. The type field specifies an XSD datatype, and the value specifies an XML instance of an object of the specified type.

Figure 18: Messages

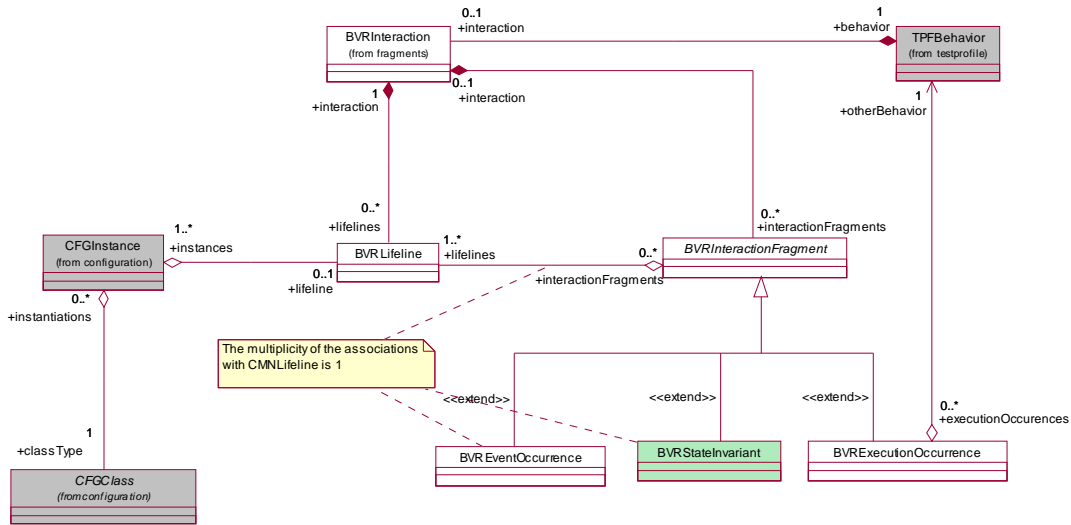


Figure 19: Lifelines

4.8.2 Classes

4.8.2.1 BVRLifeline (NormalClass)

A lifeline represents an individual participant in the Interaction. Participants represents instances of Deployable.

4.8.2.2 BVRInteractionFragment (NormalClass)

InteractionFragment is an abstract notion of the most general interaction unit. An interaction fragment is a piece of an interaction. Each interaction fragment is conceptually like an interaction by itself.

The containers of an InteractionFragment are:

- Behavior
- CombinedFragment
- InteractionOperand

4.8.2.3 BVRMessage (NormalClass)

A Message defines a particular communication between Lifelines of an Interaction.

A Message defines one specific kind of communication in an Interaction. A communication can be e.g. raising a signal, invoking an operation, creating or destroying an instance. The Message specifies not only the kind of communication given by the dispatching ExecutionOccurrence, but also the sender and the receiver.

A Message associates normally two EventOccurrences - one sending EventOccurrence and one receiving Event-Occurrence.

4.8.2.4 BVRGeneralOrdering (NormalClass)

A GeneralOrdering represents a binary relation between two Eventoccurrences, to describe that one Eventoccurrence must occur before the other.

This allows for instance the synchronization between two test components.

4.8.2.5 BVREventOccurrence (NormalClass)

EventOccurrences represents moments in time to which Actions are associated. An EventOccurrence is the basic semantic unit of Interactions. The sequences of Eventoccurrences are the meanings of Interactions. Messages are sent through either asynchronous signal sending or operation calls. Likewise they are recieved by Receptions or actions of consumption.

4.8.2.6 BVRMessageEnd (NormalClass)

A MessageEnd is an abstract concept that represents what can occur at the end of a Message.

4.8.2.7 BVRMessageSort (NormalClass)

The MessageSort captures the different kind of messages that can be used in an Interaction.

4.8.2.8 BVRExecutionOccurrence (NormalClass)

An ExecutionOccurrence is an instantiation of a unit of behavior within the Lifeline. An example of such a behavior would be the invocation of a pre-existing behavior of the DeployableInstance to which the Lifeline belongs.

Since the ExecutionOccurrence will have some duration, it is represented by two Eventoccurrences, the start EventOccurrence and the finish Event-Occurrence.

4.8.2.9 BVRStateInvariant (NormalClass)

A StateInvariant is a constraint on the state of a Lifeline. In this case we mean by state also the values of eventual attributes of the DeployableInstance the Lifeline refers to.

4.8.2.10 BVRStop (NormalClass)

A Stop is an EventOccurrence that defines the termination of the instance specified by the Lifeline on which the Stop occurs

4.8.2.11 BVRProperty (NormalClass)

4.9 fragments

4.9.1 Overview

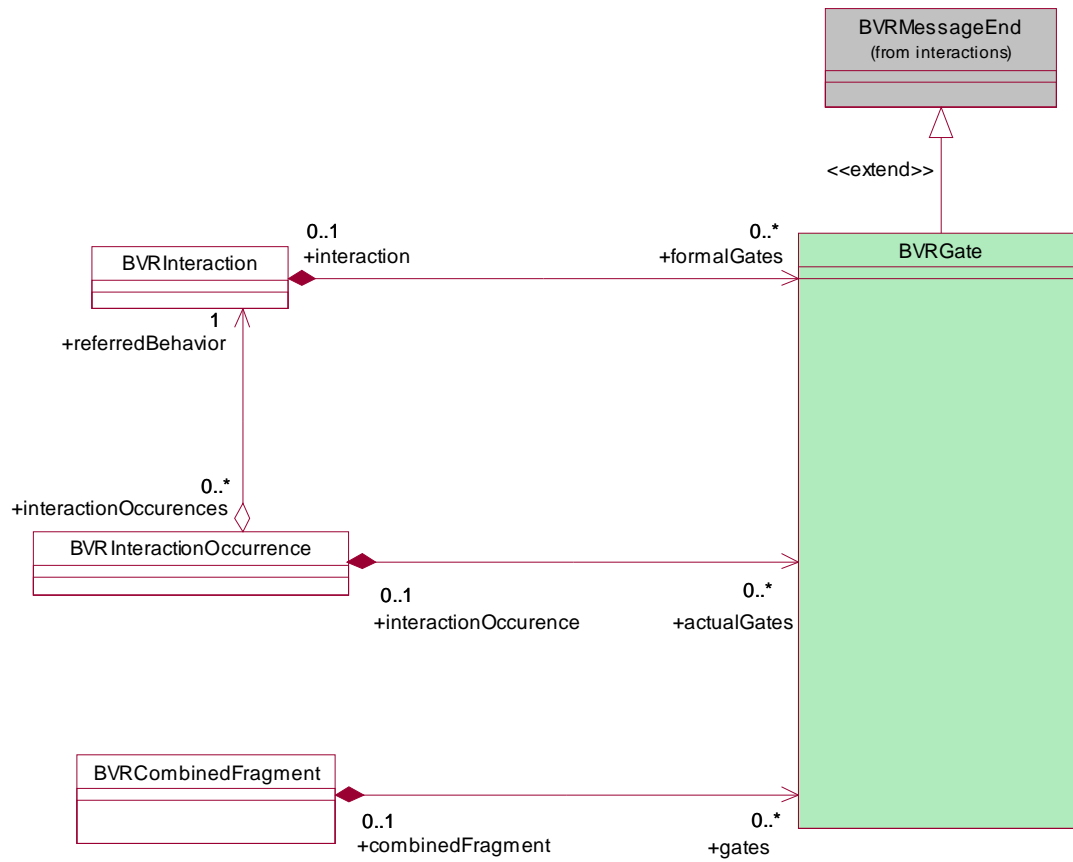


Figure 20: Gates

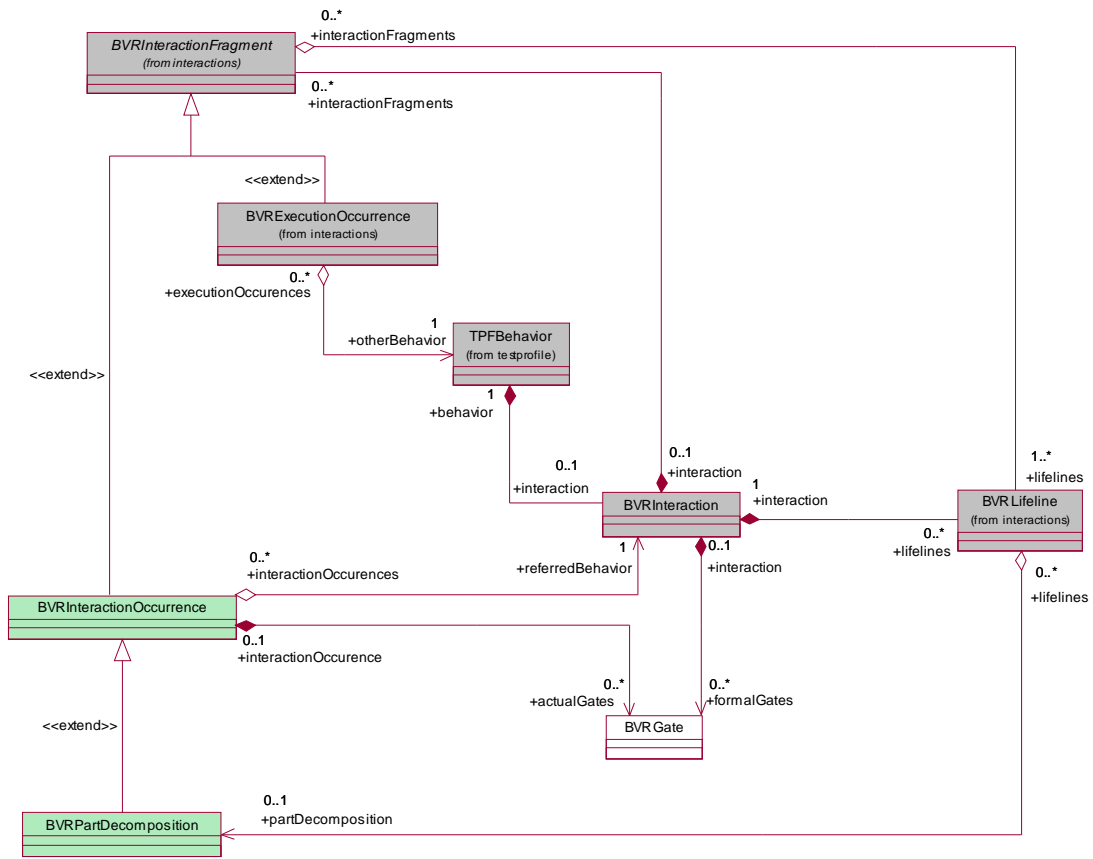


Figure 21: InteractionOccurrences

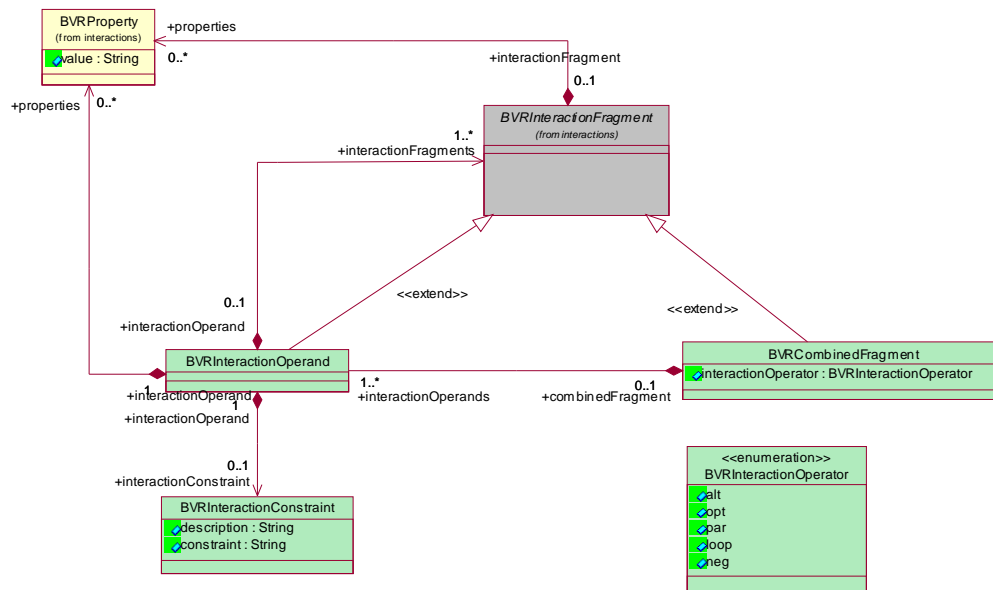


Figure 22: CombinedFragments

4.9.2 Classes

4.9.2.1 BVRInteractionOccurrence (NormalClass)

An InteractionOccurrence refers to an Interaction. The InteractionOccurrence is a shorthand for copying the contents of the referred Interaction where the InteractionOccurrence is.

It is common to want to share portions of an interaction between several other interactions. An InteractionOccurrence allows multiple interactions to reference an interaction that represents a common portion of their specification.

4.9.2.2 BVRPartDecomposition (NormalClass)

PartDecomposition is a description of the internal interactions of one Lifeline relative to an Interaction.

A Lifeline represents a DeployableInstance. A DeployableInstance is compositional and therefore its constituents interactions might be described using a PartDecomposition.

4.9.2.3 BVRInteractionOperand (NormalClass)

An InteractionOperand is contained in a CombinedFragment. An InteractionOperand represent one operand of the expression given by the enclosing CombinedFragment.

An InteractionOperand is an InteractionFragment with an optional guard expression. An InteractionOperand may be guarded by a InteractionConstraint. Only InteractionOperands with a guard that evaluates to true at this point in the interaction will be considered for the enclosing CombinedFragment.

4.9.2.4 BVRInteractionConstraint (NormalClass)

An InteractionConstraint is a boolean expression that guards an operand in a CombinedFragment.

4.9.2.5 BVRInteractionOperator (NormalClass)

The InteractionOperator is an enumerated type holding the different kinds of operators to be used in a Combined Fragment.

- alt designates that the CombinedFragment represents a choice of behavior.
- opt designates that the CombinedFragment represents a choice of behavior where either the (sole) operand happens or nothing happens.
- par designates that the CombinedFragment represents a parallel merge between the behaviors of the operands. The eventoccurrences of the different operands can be interleaved in any way as long as the ordering imposed by each operand as such is preserved.
- loop designates that the CombinedFragment represents a loop. The loop operand will be repeated a number of times.
- neg designates that the CombinedFragment represents traces that are defined to be invalid.

4.9.2.6 BVRGate (NormalClass)

A Gate is a connection point for relating a Message outside an InteractionFragment with a Message inside the

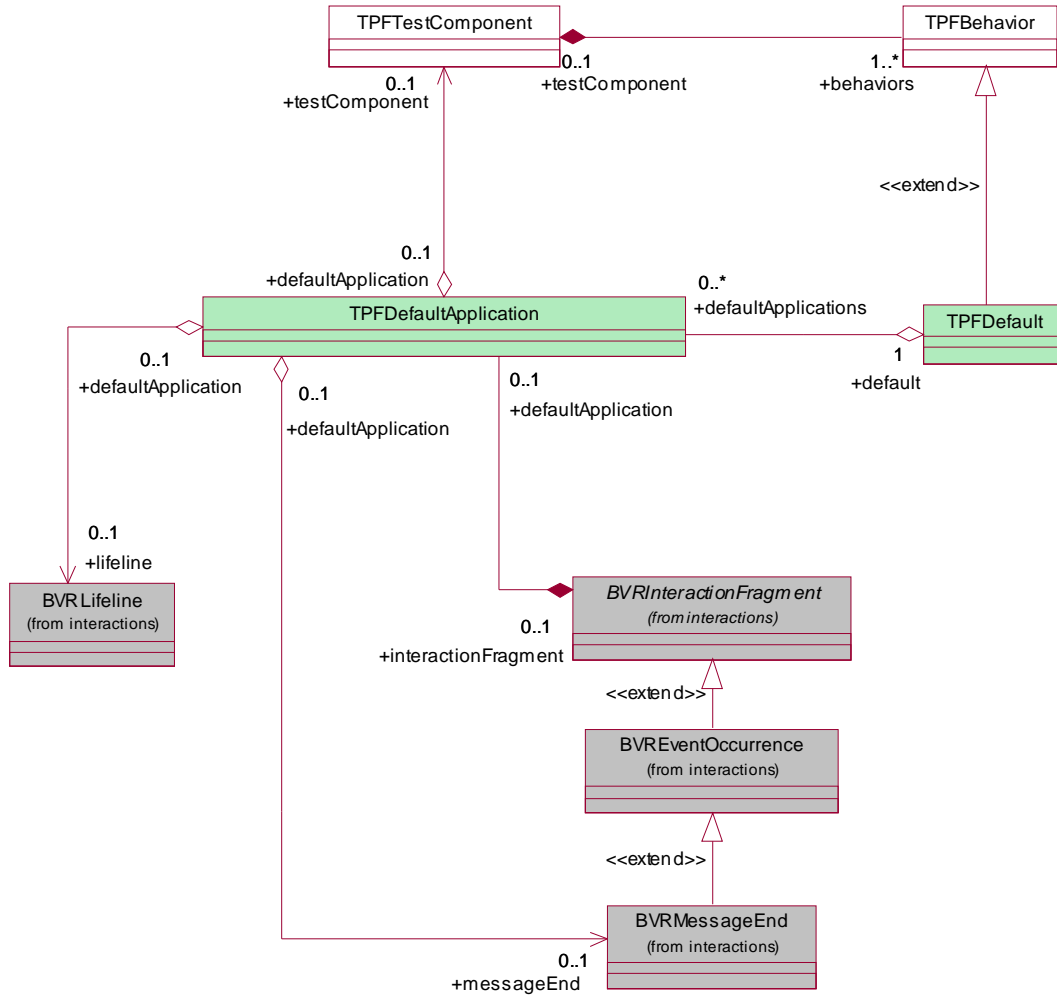


Figure 24: Defaults

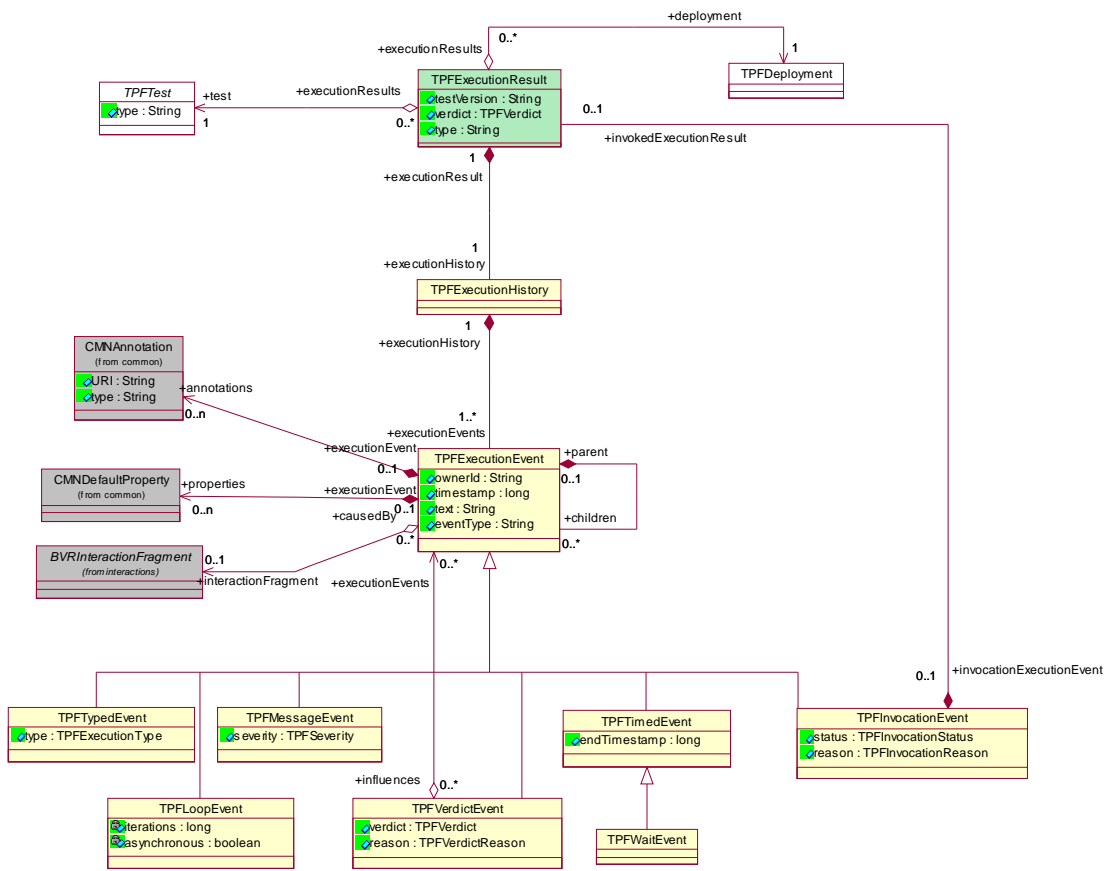


Figure 25: Execution

Each TPVerdictEvent Specifies a TPVerdict and a TPVerdictReason.

<<enumeration>>
TPVerdict

- inconclusive : String
- pass : String
- fail : String
- error : String

<<enumeration>>
TPVerdictReason

- unknown : String
- none : String
- seeDescription : String
- abort : String
- didNotComplete : String

Each TPInvocationEvent specifies a TPInvocationStatus and a TPInvocationReason

<<enumeration>>
TPInvocationStatus

- unattempted : String
- successful : String
- unsuccessful : String

<<enumeration>>
TPInvocationReason

- unknown : String
- none : String
- seeDescription : String
- preconditionNotMet : String
- noMatchingConfiguration : String
- noBehavior : String
- didNotStart : String

Each TPTypeEvent specifies a TPExecutionType

<<enumeration>>
TPExecutionType

- start : String
- stop : String

Each TPMessageEvent specifies a TPSeverity

<<enumeration>>
TPSeverity

- info : String
- error : String
- warning : String

Figure 26: Execution Enumerations

4.10.2 Classes

4.10.2.1 TPFDeployment (NormalClass)

A deployment is the allocation of the Instances into Locations.

A Test Suite might be associated to a number of deployment specifying with configurations should be used to run the Test Suite.

4.10.2.2 TPFLiteralAnyOrNull (NormalClass)

LiteralAnyOrNull is a literal representing any value out of a set of possible values, or the lack of a value.

4.10.2.3 TPFLiteralAny (NormalClass)

LiteralAny is a literal representing any value out of a set of possible values

4.10.2.4 TPFInstanceValue (NormalClass)

An Instance Value is a specification of a named attribute value. The value can take several forms including literal values, expressions, existing InstanceValues, or constraints.

4.10.2.5 TPFExecutionResult (NormalClass)

The ExecutionResult (renamed Trace concept from the Test Profile) represents a snapshot of the execution of a TestCase or TestSuite. It contains deployment information, the ordered set of log actions performed during the test case, and the final verdict.

4.10.2.6 TPFLogAction (NormalClass)

A LogAction is an EventOccurrence that specifies that an entity should be logged to the execution trace for further analysis. The target of a log action is a logging mechanism in the run-time system.

4.10.2.7 TPFDefault (NormalClass)

A default is a behavior that is activated when an unexpected event occurs on a test component.

4.10.2.8 TPFDefaultApplication (NormalClass)

A default application is the a reference to a default behavior, owned by an element defining its scope. This allows the default to be activated should an unexpected event occur on a component, based on its scope.

4.10.2.9 TPFBehavior (NormalClass)

Behavior represents the dynamic behavior of a TestSuite, TestCase, TestComponent used to test an SUT, or Arbiter. In the context of a TestCase, it represents the composite behavior of the different TestComponents realizing the test, and/or the behavior of the TestCase in the case where the TestSuite classifier interacts directly with the SUT.

In the Hyades meta-model Behaviors are modeled using Interactions (i.e. Sequence Diagrams) enabling the easy modeling of composite behaviors.

4.10.2.10 TPFTestCase (NormalClass)

A test case is a specification of one case to test the system, including what to test with which input, result, and under which conditions.

A test case belongs to a test suite. It has a behavior specifying how a set of cooperating test components interacting with a system under test realize a test objective.

It may invoke other test cases.

A test case uses an arbiter to evaluate the outcome of its test behavior.

4.10.2.11 TPFArbiter (NormalClass)

An Arbiter represents an implementation of the IArbiter interface. The purpose of an IArbiter implementation is to determine the final verdict for a test case. This determination is done according to a particular arbitration strategy, which is provided in the implementation of the arbiter interface.

4.10.2.12 TPFCodingRule (NormalClass)

Coding rules are strings referencing coding rules such as those defined for ASN.1, CORBA or XML. Coding rules are basically applied to InstanceValue to denote the concrete encoding and decoding for these values during test execution.

4.10.2.13 TPFSUT (NormalClass)

The SUT is the system under test. The SUT provides only a set of operations via publicly available interface(s) to enable blackbox testing.

4.10.2.14 TPFTestSuite (NormalClass)

A Test Suite acts as a grouping mechanism for a set of test cases.

It can be related to a TestObjective, defining the motivation of the TestSuite.

The behavior of a test suite is generally used for control the execution flow between its test cases or to invokes external test suites or test cases.

4.10.2.15 TPFTestObjective (NormalClass)

A Test Objective is a dependency used to specify the objectives of a Test Case or Test Suite. A Test Case or Test Suite can have any number of objectives. A Test Objective being a dependency to the objective itself, it is associated to only one Test Case or Test Suite.

4.10.2.16 TPFTestComponent (NormalClass)

A test component is commonly an active class used to specify test cases as interactions between a number of test components with the SUT. The behavior of a test component can be used to specify low level test behavior, or it can be automatically derived from all the test cases behaviors in which the component takes part.

4.10.2.17 TPFTimezone (NormalClass)

Timezones serve as a grouping mechanisms for test components within a test system. Each test component belongs at most to one timezone. Test components in the same timezone have the same perception of time, i.e. test components of the same timezone are considered to be time synchronized. The time zone of a test component can be accessed both in the model and in run-time.

Comparing time-critical events within the same timezone is allowed. Comparing time-critical events of different timezones is a matter of semantic variation point and should be decided by the tool vendor. By default, comparison between events in two different timezones is illegal.

4.10.2.18 TPFValidationAction (NormalClass)

A ValidationAction is an EventOccurrence that performs some kind of verification.

When a validation action is specified, it indicates that at run-time, the expression within the action will be evaluated. If the expression evaluates to true, the verdict pass is sent to the arbiter using the setVerdict operation. If the expression is false, the verdict fail is sent to the arbiter. Instead of an expression, valid verdicts for the arbiter implementation may also be used.

4.10.2.19 TPFVerdict (NormalClass)

The verdict is a predefined enumeration datatype which contains at least the values fail, inconclusive, pass, error indicating how this test case execution has performed.

- A pass verdict indicates that the test case is successful and that the SUT has behaved according to what should be expected.
- A fail verdict on the other hand shows that the SUT is not behaving according to the specification.
- An inconclusive verdict means that the test execution cannot determine whether the SUT performs well or not.
- An error verdict tells that the test system itself and not the SUT fails.

4.10.2.20 TPFExecutionHistory (NormalClass)

4.10.2.21 TPFExecutionEvent (NormalClass)

4.10.2.22 TPFExecutionStatus (NormalClass)

4.10.2.23 TPFExecutionType (NormalClass)

4.10.2.24 TPFInvocationEvent (NormalClass)

4.10.2.25 TPFVerdictEvent (NormalClass)

4.10.2.26 TPFMessageEvent (NormalClass)

4.10.2.27 TPFSeverity (NormalClass)

4.10.2.28 TPFTypedEvent (NormalClass)

4.10.2.29 TPFInvocationStatus (NormalClass)

4.10.2.30 TPFVerdictReason (NormalClass)

4.10.2.31 TPFInvocationReason (NormalClass)

4.10.2.32 TPFTest (NormalClass)

A test is an abstract concept for TestSuite and Test Case. It enables the association of both concepts to a TestObjective, a TestExecutionResult and a Behavior.

As such a Test has no semantic associated with it.

4.10.2.33 TPFLoopEvent (NormalClass)

4.10.2.34 TPFTimedEvent (NormalClass)

A TPFTimedEvent is an event that is specifically timed. It contains both a start and end timestamp, and can have its duration calculated and displayed.

4.10.2.35 TPFWaitEvent (NormalClass)

A TPFWaitEvent is a specialization of TPFTimedEvent. The distinction between the two types is that a TPFWaitEvent is an event that logs a deliberate wait or sleep, while a TPFTimedEvent may log the duration of any arbitrary sequence.

4.11 configuration

4.11.1 Overview

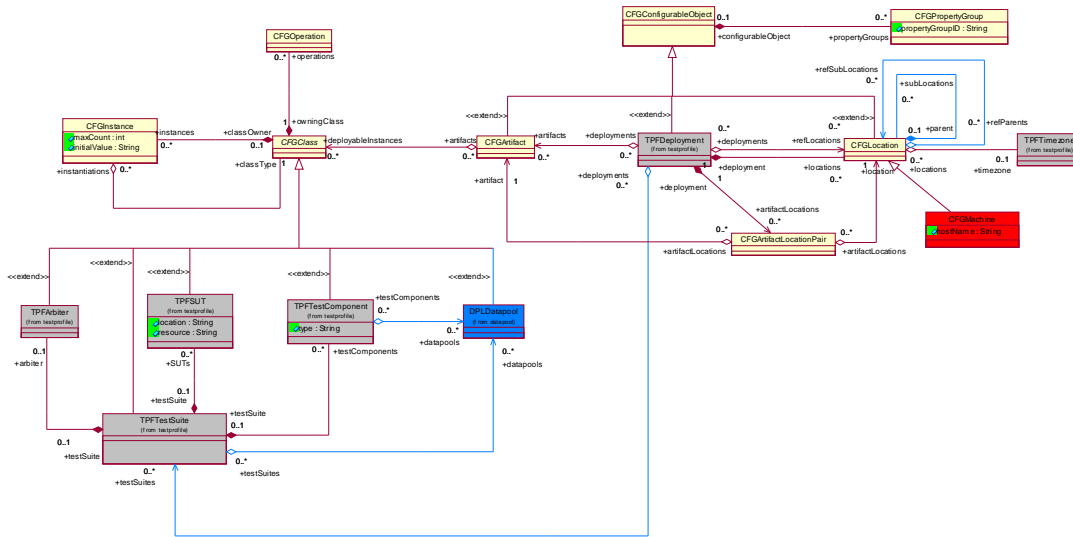
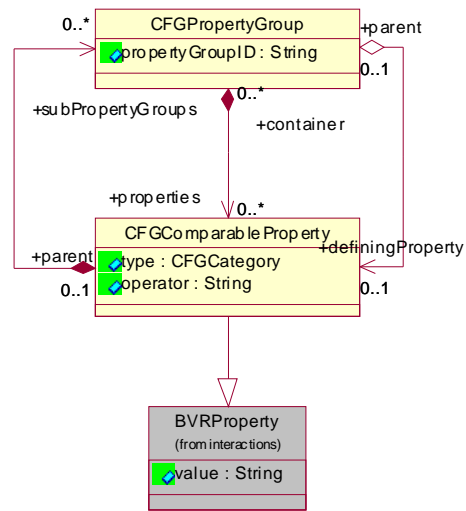


Figure 27: Main



1. Need the ability to capture state of CMN Machine
2. Need the ability to store relationships among properties (mutually exclusive, lockable by an given Agent / Test Run, etc.)
3. Can we share CFGMachineInstance with TRCNode

Figure 28: PropertyGroup

Figure 29: Category

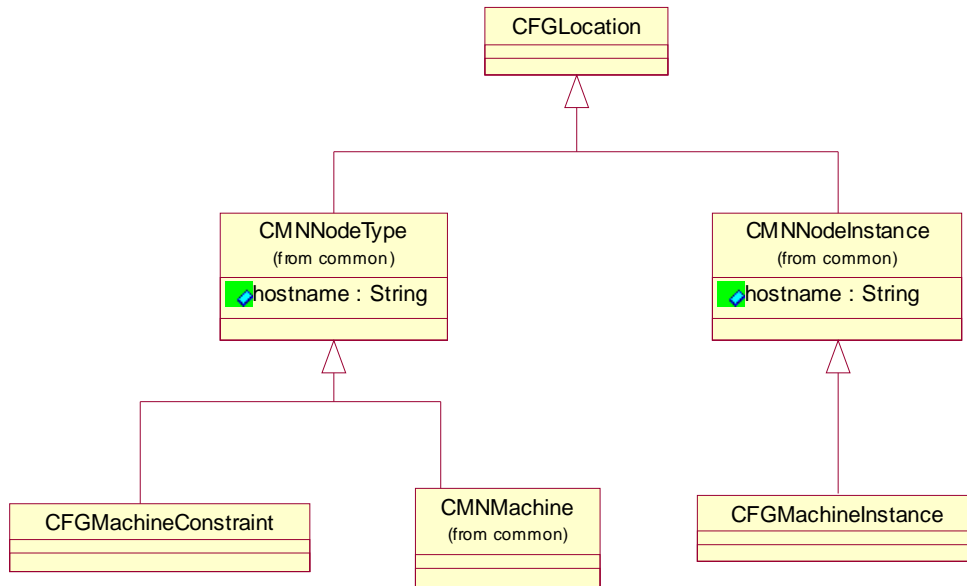


Figure 30: Location

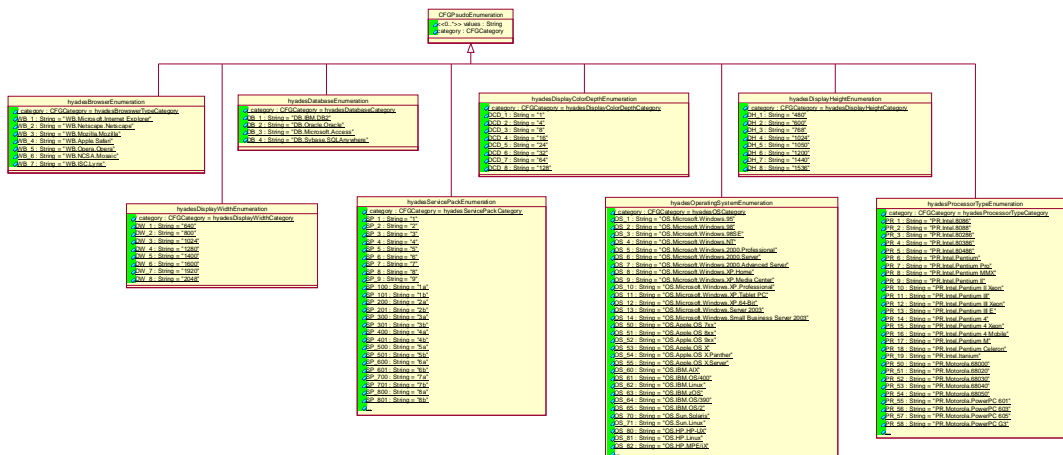


Figure 31: PseudoEnumerations

4.11.2 Classes

4.11.2.1 CFGLocation (NormalClass)

A CFGLocation is the representation of a physical or logical entity into which a CFGInstance can be deployed.

Examples of Location include Processors, Application Servers, Containers, JVM, Processes, Threads.

4.11.2.2 CFGClass (NormalClass)

A CFGDeployable is an element that describes behavioral (operation/behavior), and structural features (Instance) and may be instantiable. Mapped to a programming language like java, it comes in several specific forms, including class, interface, etc.

It owns CFGInstances which define its internal structure (class attributes).

A CFGDeployable has a behaviorResource and behaviorLocation which describe where its behavior is stored: Resource URL and Location within the resource .

4.11.2.3 CFGOperation (NormalClass)

An operation is a service that can be requested from a deployable.

4.11.2.4 CFGArtifact (NormalClass)

This class is a grouping of CFGInstances that share a common CFGDeploymentSpec and CFGLocation. In other words, CFGInstances within the same CFGArtifact will be deployed to the same location, and within the same {process / thread}.

4.11.2.5 CFGInstance (NormalClass)

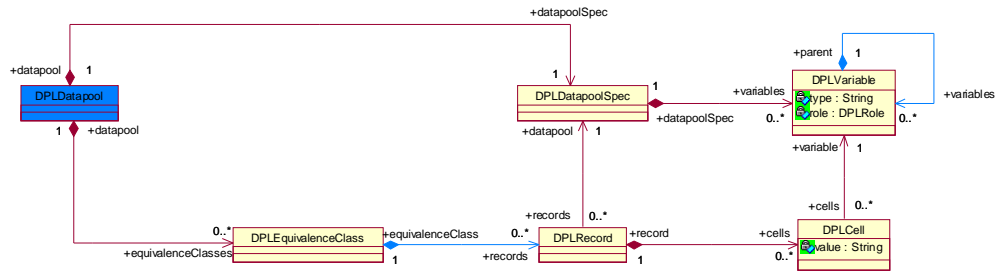
CFGInstance represents an instance variable of a CFGClass. The objects represented by these instance variables are grouped in one or more artifacts. The maxCount attribute of CFGInstance represents the maximum number of instances of the object represented by this instance variable which will be created during a test execution.

- 4.11.2.6 [CFGComparableProperty \(NormalClass\)](#)
- 4.11.2.7 [CFGPropertyGroup \(NormalClass\)](#)
- 4.11.2.8 [CFGCategory \(NormalClass\)](#)
- 4.11.2.9 [hyadesOperatingSystemEnumeration \(NormalClass\)](#)
- 4.11.2.10 [hyadesOperatingSystemCategory \(NormalClass\)](#)
- 4.11.2.11 [hyadesProcessorTypeCategory \(NormalClass\)](#)
- 4.11.2.12 [hyadesMemorySizeCategory \(NormalClass\)](#)
- 4.11.2.13 [CFGCategorySelectionMode \(NormalClass\)](#)
- 4.11.2.14 [hyadesProcessorSpeedCategory \(NormalClass\)](#)
- 4.11.2.15 [hyadesProcessorTypeEnumeration \(NormalClass\)](#)
- 4.11.2.16 [hyadesBrowserTypeCategory \(NormalClass\)](#)
- 4.11.2.17 [hyadesBrowserVersionCategory \(NormalClass\)](#)
- 4.11.2.18 [hyadesBrowserEnumeration \(NormalClass\)](#)
- 4.11.2.19 [hyadesProcessorNumberCategory \(NormalClass\)](#)
- 4.11.2.20 [hyadesDisplayColorDepthCategory \(NormalClass\)](#)
- 4.11.2.21 [hyadesDisplayColorDepthEnumeration \(NormalClass\)](#)
- 4.11.2.22 [hyadesDisplayHeightCategory \(NormalClass\)](#)
- 4.11.2.23 [hyadesDisplayWidthCategory \(NormalClass\)](#)
- 4.11.2.24 [hyadesDisplayHeightEnumeration \(NormalClass\)](#)
- 4.11.2.25 [hyadesDisplayWidthEnumeration \(NormalClass\)](#)
- 4.11.2.26 [hyadesDisplayNumberCategory \(NormalClass\)](#)
- 4.11.2.27 [hyadesDatabaseCategory \(NormalClass\)](#)

- 4.11.2.28 [hyadesDatabaseVersionCategory \(NormalClass\)](#)
- 4.11.2.29 [hyadesDatabaseEnumeration \(NormalClass\)](#)
- 4.11.2.30 [CFGMachineInstance \(NormalClass\)](#)
- 4.11.2.31 [CFGMachineConstraint \(NormalClass\)](#)
- 4.11.2.32 [CFGCategorySelectionAmount \(NormalClass\)](#)
- 4.11.2.33 [CFGCategorySelectionMultiplicity \(NormalClass\)](#)
- 4.11.2.34 [CFGConfigurableObject \(NormalClass\)](#)
- 4.11.2.35 [hyadesHostnameCategory \(NormalClass\)](#)
- 4.11.2.36 [hyadesWindowsDomainCategory \(NormalClass\)](#)
- 4.11.2.37 [hyadesUsernameCategory \(NormalClass\)](#)
- 4.11.2.38 [hyadesServicePackCategory \(NormalClass\)](#)
- 4.11.2.39 [hyadesMajorVersionCategory \(NormalClass\)](#)
- 4.11.2.40 [hyadesMinorVersionCategory \(NormalClass\)](#)
- 4.11.2.41 [hyadesServicePackEnumeration \(NormalClass\)](#)
- 4.11.2.42 [CFGArtifactLocationPair \(NormalClass\)](#)
- 4.11.2.43 [CFGMachine \(NormalClass\)](#)
- 4.11.2.44 [CFGPseudoEnumeration \(NormalClass\)](#)
- 4.11.2.45 [hyadesPasswordCategory \(NormalClass\)](#)
- 4.11.2.46 [hyadesClasspathCategory \(NormalClass\)](#)
- 4.11.2.47 [hyadesRootDirectroyCategory \(NormalClass\)](#)

4.12 *datapool*

- 4.12.1 Overview



UML 2.0 Mapping
 DPLDatapool = Classifier
 DPLDatapoolSpec = Classifier
 DPLRecord = InstanceSpec
 DPLEquivalenceClass = InstanceSpec
 DPLCell = Slot
 DPLVariable = StructuredFeature

```

<<enumeration>>
DPLRole
+ InspectedData : String
+ InputData : String
+ OutputData : String
+ InputOutputData : String
  
```

Figure 32: Main

4.12.2 Classes

4.12.2.1 DPLDatapoolSpec (NormalClass)

4.12.2.2 DPLEquivalenceClass (NormalClass)

4.12.2.3 DPLRecord (NormalClass)

4.12.2.4 DPLCell (NormalClass)

4.12.2.5 DPLVariable (NormalClass)

4.12.2.6 DPLRole (NormalClass)

4.12.2.7 DPLDatapool (NormalClass)

4.13 *trace*

4.13.1 Overview

The purpose of the trace model is to capture execution behavior of an application. This includes call stack tracing as well as object/memory observation. All over multi process/tier environments.

TRCAggregatedObject Reference
ownerSize : int
targetSize : int
count : int

<<enumeration>> TRCPrimitiveType
JAVA_REFERENCE = 1
JAVA_BOOLEAN
JAVA_BYTE
JAVA_CHAR
JAVA_SHORT
JAVA_INT
JAVA_LONG
JAVA_FLOAT
JAVA_DOUBLE

<<datatype>> EObjectID
<<javaclass>> long

<<datatype>> EMethodID
<<javaclass>> int

<<datatype>> EClassID
<<javaclass>> int

<<enumeration>> TRCSignatureNotation
JNI = 0

<<enumeration>> TRCMethodProperties
JAVA_NATIVE = 1
JAVA_SYNCHRONIZED = 2
JAVA_PUBLIC = 4
JAVA_PRIVATE = 8
JAVA_PROTECTED = 16
JAVA_DEFAULT_VISIBILITY = 32
JAVA_STATIC = 64
JAVA_CONSTRUCTOR = 128
JAVA_ABSTRACT = 256
JAVA_FINALIZER = 512

<<enumeration>> TRCGCRootType
UNKNOWN = -1
JNI_GLOBAL
JNI_LOCAL
JAVA_FRAME
NATIVE_STACK
STICKY_CLASS
THREAD_BLOCK
MONITOR_USED

Figure 33: Basic Properties

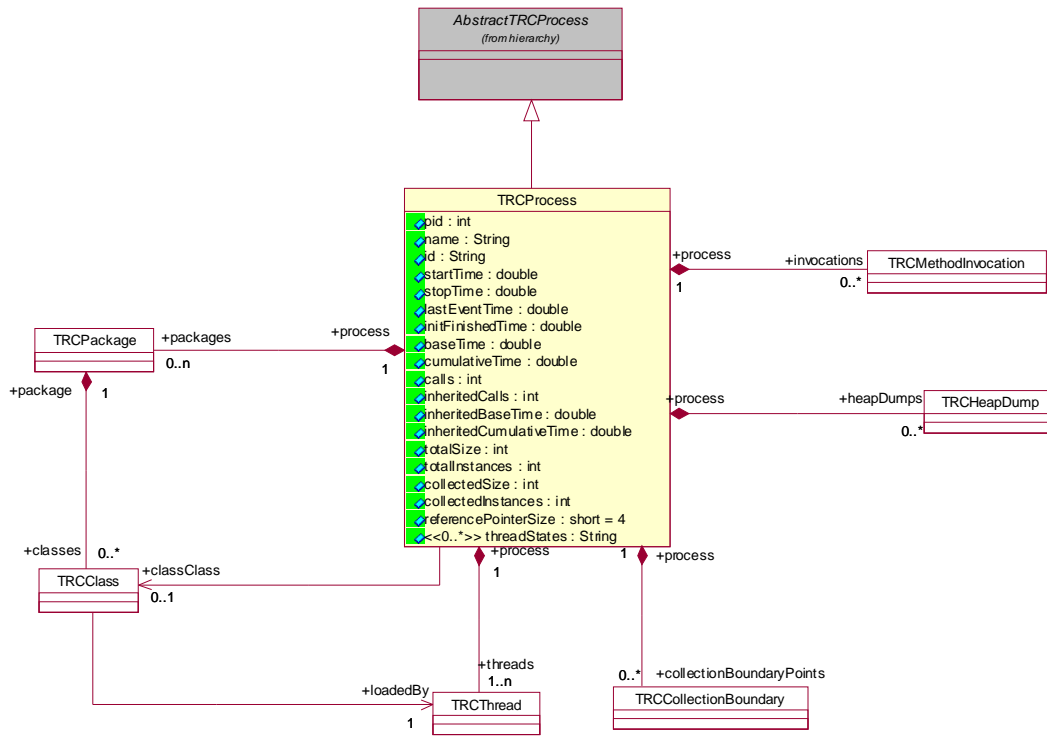


Figure 34: Process

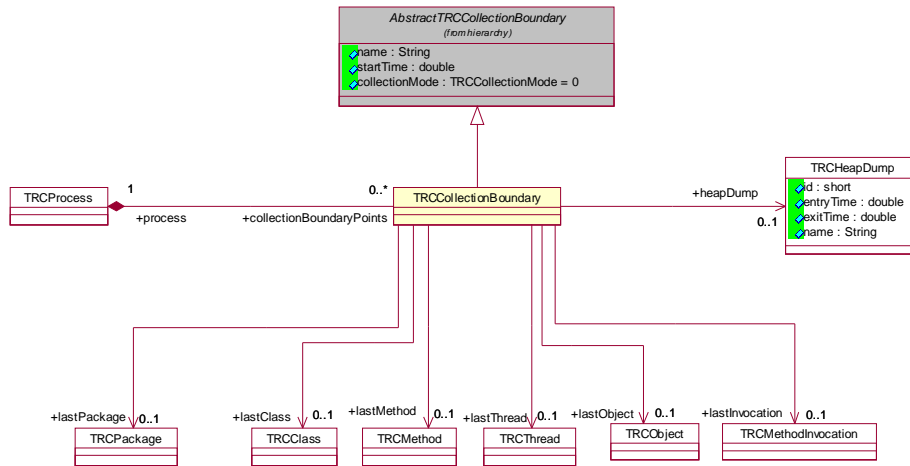


Figure 35: Collection boundaries

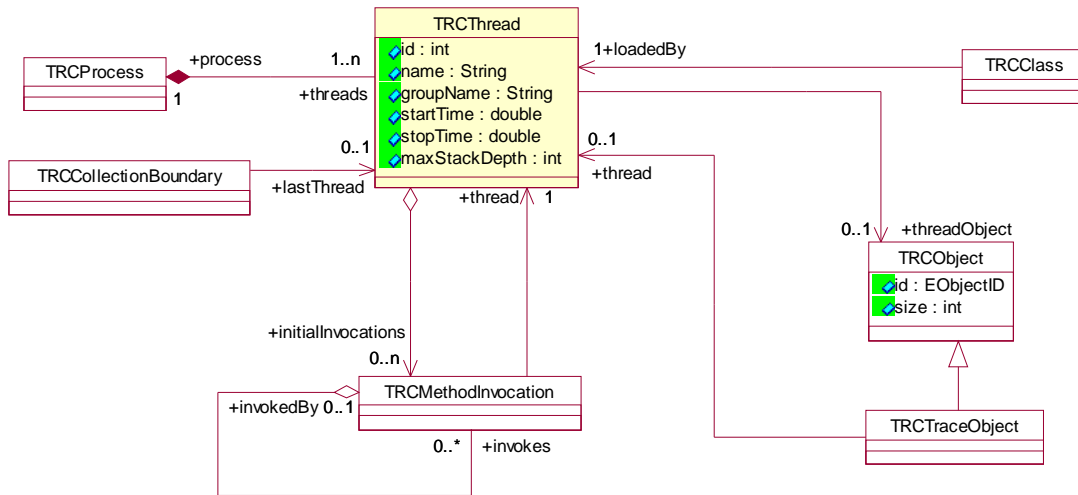


Figure 36: Thread

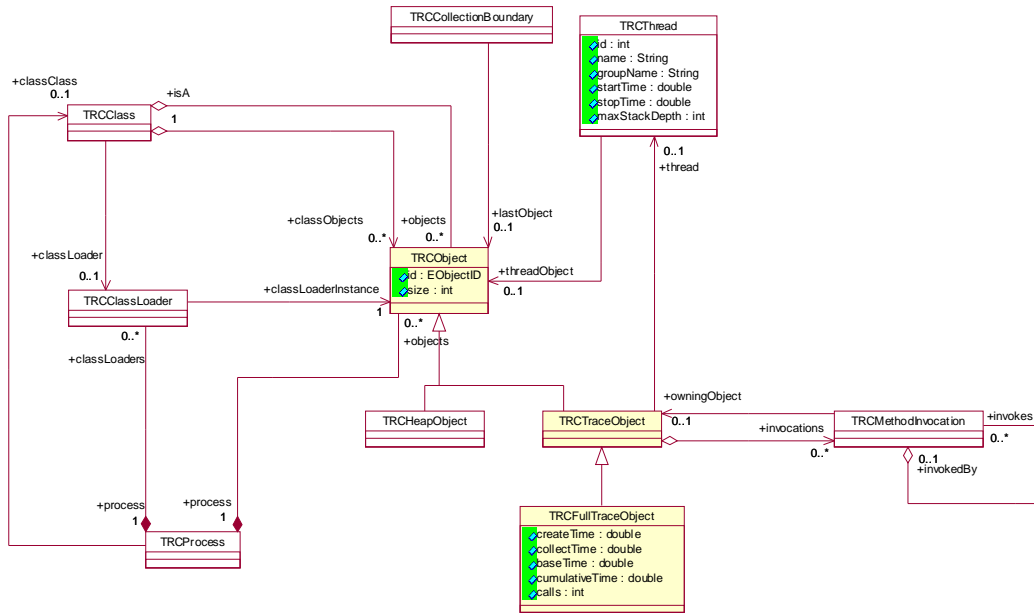


Figure 37: Execution Object Aspect

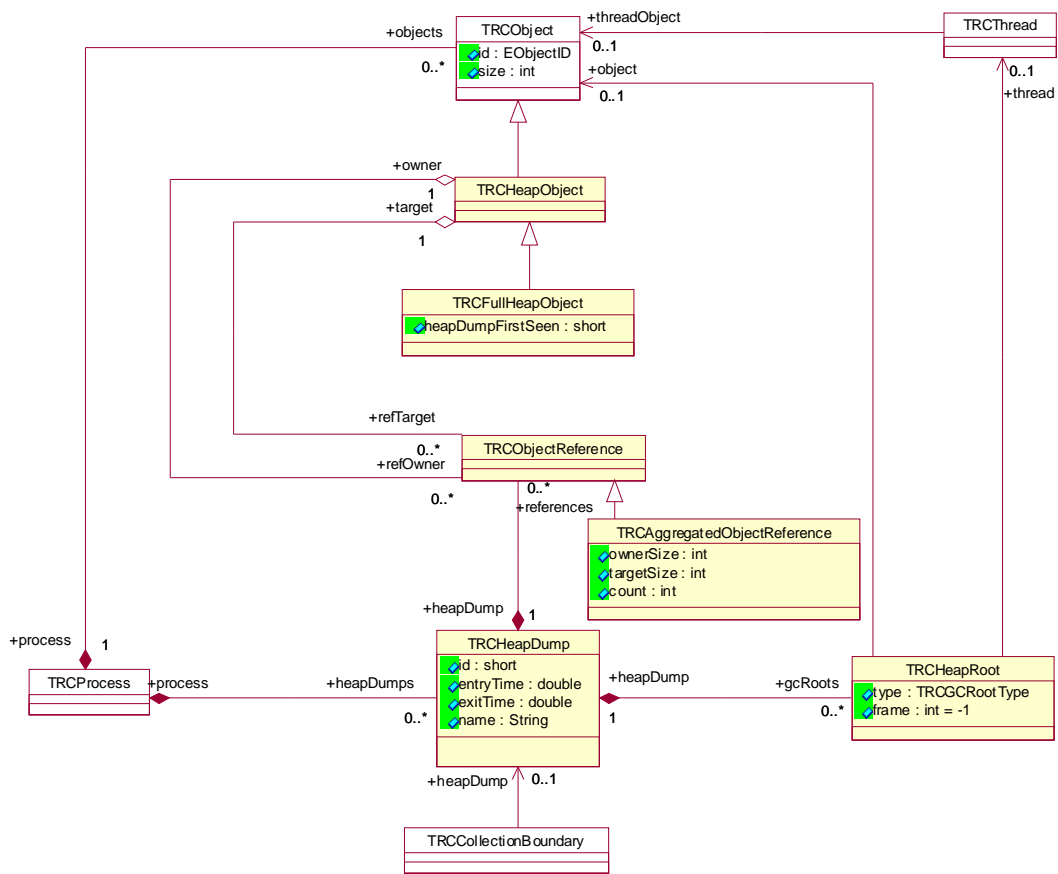


Figure 38: Heap dump aspect

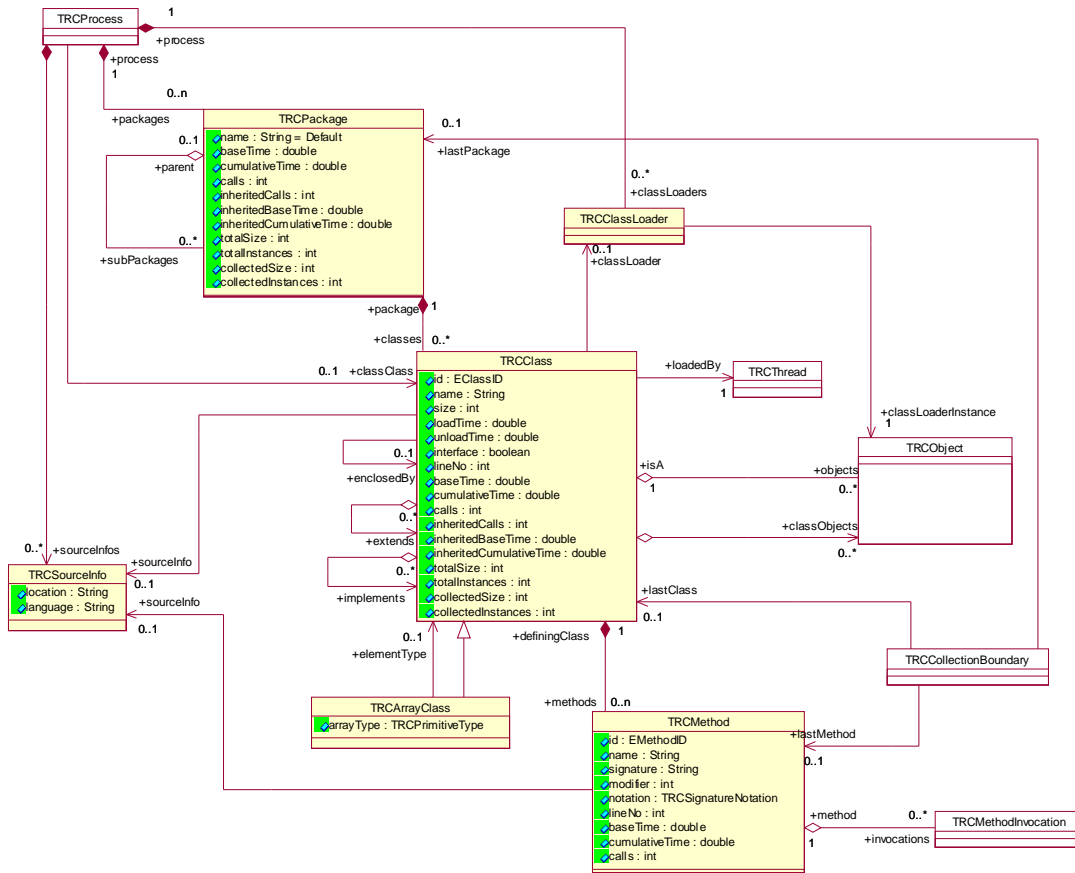


Figure 39: Definition Aspect

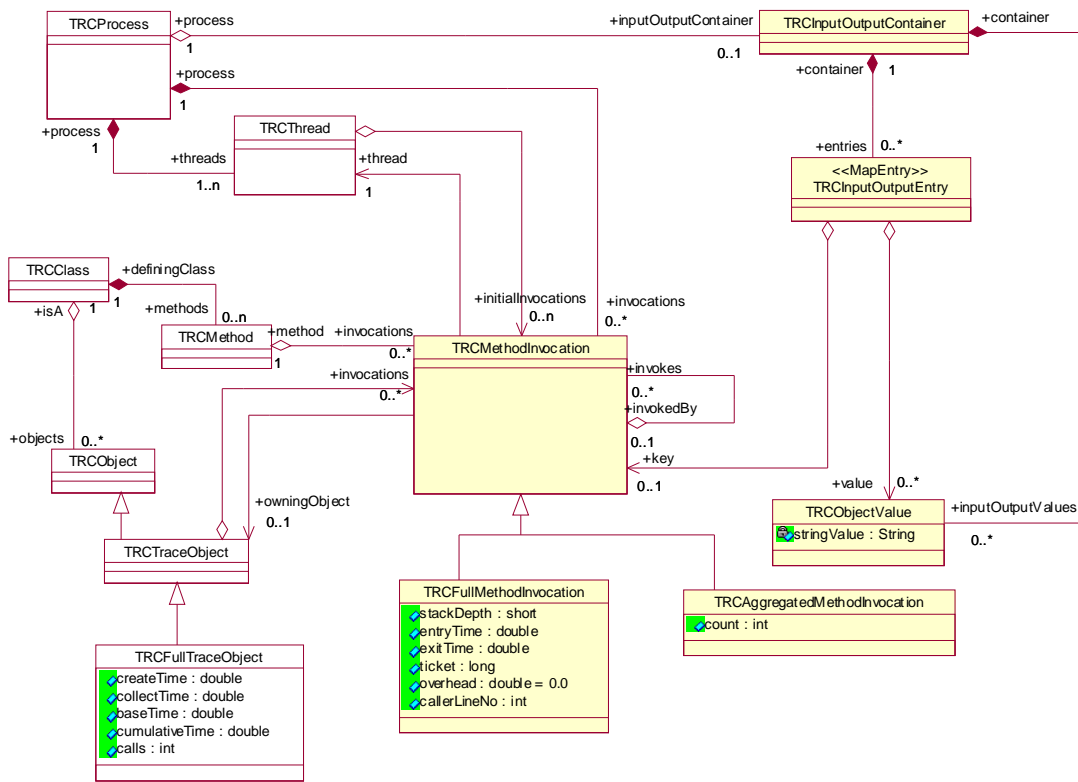


Figure 40: Execution Aspect

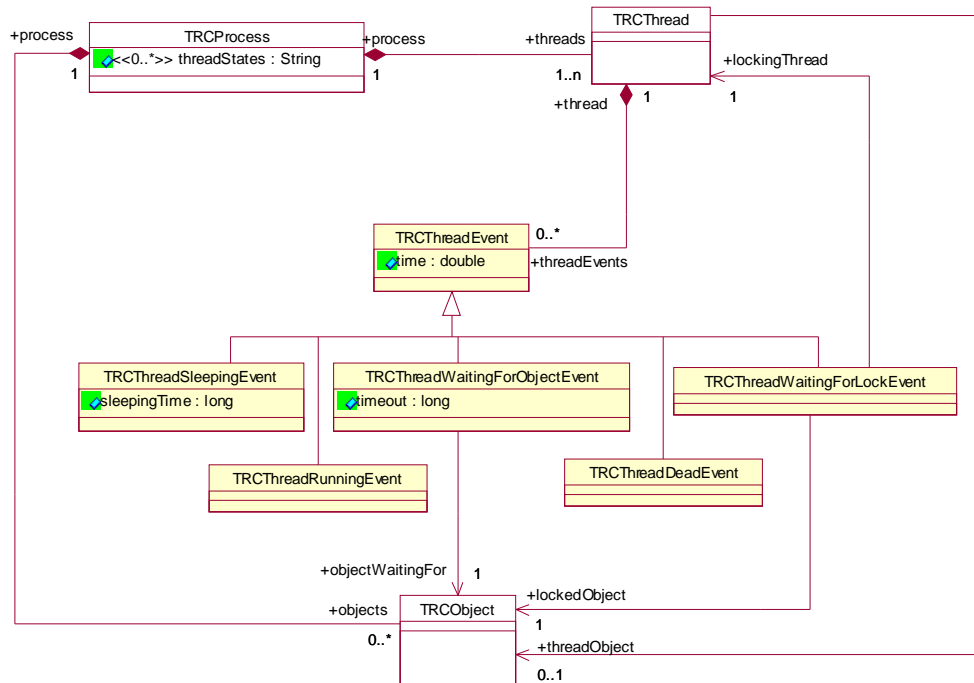


Figure 41: Thread events

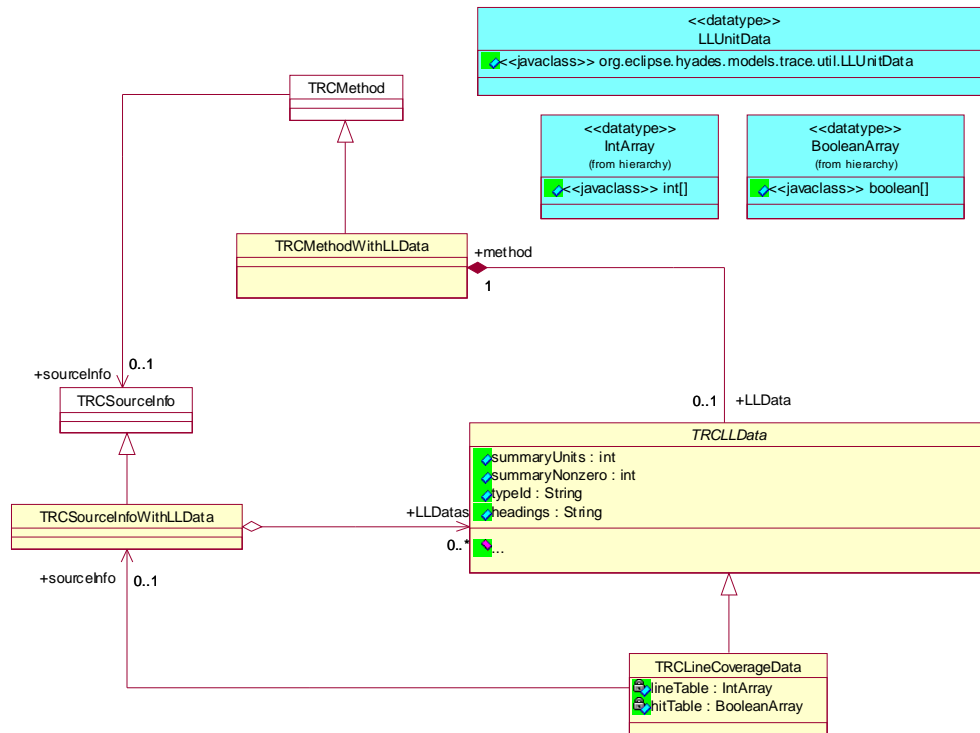


Figure 42: Line Level Data

4.13.2 Classes

4.13.2.1 TRCObject (NormalClass)

Represents an instance of a type (defined by TRCClass)

4.13.2.2 TRCClass (NormalClass)

This represents a complex type (a Java class for example)

- calls, baseTime, and cumulativeTime: these are aggregate measures of all the method invocations whose receiver object is of the given class (or package, or process). For cumulativeTime, time spent in any method invocations that call each other is not counted more than once for that class (or package, or process).

- inheritedCalls, inheritedBaseTime, and inheritedCumulativeTime: these are aggregate measures of all the method invocations whose receiver object is of the given class (or package, or process), but limited to those whose method is not implemented in the class of the receiver object (i.e. it is implemented in a class further up in the class hierarchy).

4.13.2.3 TRCMethodInvocation (NormalClass)

A method invocation is the entry and exit of the actual method implementation.

The base class does not track any attributes, for minimum overhead.

4.13.2.4 TRCProcess (NormalClass)

TRCProcess represents a process. In this case it is one that has been monitored. It is the root container of a profiling trace.

- calls, baseTime, and cumulativeTime: these are aggregate measures of all the method invocations whose receiver object is of the given class (or package, or process). For cumulativeTime, time spent in any method invocations that call each other is not counted more than once for that class (or package, or process).

- inheritedCalls, inheritedBaseTime, and inheritedCumulativeTime: these are aggregate measures of all the method invocations whose receiver object is of the given class (or package, or process), but limited to those whose method is not implemented in the class of the receiver object (i.e. it is implemented in a class further up in the class hierarchy).

4.13.2.5 TRCThread (NormalClass)

A monitored thread, scoped by its owning process.

4.13.2.6 TRCMethod (NormalClass)

A method definition

4.13.2.7 TRCPrimitiveType (NormalClass)

4.13.2.8 EObjectID (NormalClass)

Custom EMF EDataType to support conversion from a special object ID format to long.

4.13.2.9 EMethodID (NormalClass)

Custom EMF EDataType to support conversion from a special method ID format to int.

4.13.2.10 TRCPackage (NormalClass)

Logical representation of a Java package

- calls, baseTime, and cumulativeTime: these are aggregate measures of all the method invocations whose receiver object is of the given class (or package, or process). For cumulativeTime, time spent in any method invocations that call each other is not counted more than once for that class (or package, or process).

- inheritedCalls, inheritedBaseTime, and inheritedCumulativeTime: these are aggregate measures of all the method invocations whose receiver object is of the given class (or package, or process), but limited to those whose method is not implemented in the class of the receiver object (i.e. it is implemented in a class further up in the class hierarchy).

4.13.2.11 TRCCollectionBoundary (NormalClass)

A collection boundary (segment boundary) of a trace. This class marks the beginning of a segment. It can have an method invocation associated with it (the first method invocation that was received in that slice). The end of a segment is marked by the end of the trace or by the next segment.

4.13.2.12 TRCClassLoader (NormalClass)

Defines a class loader

4.13.2.13 EClassID (NormalClass)

Custom EMF EDataType to support conversion from a special class ID format to int.

4.13.2.14 TRCSignatureNotation (NormalClass)

4.13.2.15 TRCSourceInfo (NormalClass)

This is an anchor point to capture the information about a specific piece of source relative to a specific file.

4.13.2.16 TRCMethodProperties (NormalClass)

Used as a bitmask for the TRCMethod.modifier field.

4.13.2.17 TRCHeapObject (NormalClass)

4.13.2.18 TRCFullTraceObject (NormalClass)

4.13.2.19 TRCTraceObject (NormalClass)

4.13.2.20 TRCFullHeapObject (NormalClass)

4.13.2.21 TRCObjectReference (NormalClass)

4.13.2.22 TRCHeapDump (NormalClass)

4.13.2.23 TRCAggregatedMethodInvocation (NormalClass)

The TRCAggregatedMethodInvocation represents a collapsed view of several MethodInvocation along identical call chains, and maintains a count of how often the chain was repeated.

4.13.2.24 TRCFullMethodInvocation (NormalClass)

TRCFullMethodInvocation carries attributes about a method invocation that are used when the agent was tracking performance information about each function call.

4.13.2.25 TRCHeapRoot (NormalClass)

4.13.2.26 TRCGCRootType (NormalClass)

4.13.2.27 TRCArrayClass (NormalClass)

4.13.2.28 TRCAggregatedObjectReference (NormalClass)

This will be used in HEAP_STATISTICS* collection modes. This reference object will be linked at class object level and it will look like a class to class reference.

- 4.13.2.29 TRCThreadEvent (NormalClass)**
- 4.13.2.30 TRCThreadSleepingEvent (NormalClass)**
- 4.13.2.31 TRCThreadWaitingForObjectEvent (NormalClass)**
- 4.13.2.32 TRCThreadWaitingForLockEvent (NormalClass)**
- 4.13.2.33 TRCThreadRunningEvent (NormalClass)**
- 4.13.2.34 TRCThreadDeadEvent (NormalClass)**
- 4.13.2.35 TRCMethodWithLLData (NormalClass)**
- 4.13.2.36 TRCLLData (NormalClass)**
- 4.13.2.37 TRCSourceInfoWithLLData (NormalClass)**
- 4.13.2.38 TRCLineCoverageData (NormalClass)**
- 4.13.2.39 LLUnitData (NormalClass)**
- 4.13.2.40 TRCObjectValue (NormalClass)**

This represent the toString() value of an object. The collection agent can send a different string representation of the object if the output from toString() method is not appropriate.

Later we can extend this class with specialized value classes.

- 4.13.2.41 TRCInputOutputEntry (NormalClass)**

- 4.13.2.42 TRCInputOutputContainer (NormalClass)**

The container of all parameters/return values and also of the Map entries (key=TRCMethodInvocation, value=list of TRCObjectValue).

The instances of this class will be root objects in resources with trciovxmi extension.

4.14 *statistical*

4.14.1 Overview

Internal package, please do not extend or use.

This package is intended to describe structured and dynamically typed statistical data.

4.14.2.5 SDView (NormalClass)

The view is a named collection of windows. The view allows you to navigate the windows so that data can be propagated through the windows if that behaviour is desired.

For example data may be collected at 1 minute intervals and used to populate a window which contains the most recent 1 hour of data. There may be a second window that contains a set of data points that summarize each of the previous 23 hours plus the hour represented in the first window. A third window may reflect the data for a given week, and so on.

A view associates these windows of related data and provides a mechanism to manage the cascading of data across these windows over the data.

4.14.2.6 SDSampleWindow (NormalClass)

This is a window of data. Windows contain a fixed number of snapshots that typically represent some period of time.

This class can be specialized to describe processing of the observations for a particular use-case.

4.14.2.7 SDSnapshotObservation (NormalClass)

All observations are specializations of this class. The individual observation values reside in the specialization instances.

4.14.2.8 SDDiscreteObservation (NormalClass)

The collection of discrete values associated with a counter during a sample window.

4.14.2.9 SDContiguousObservation (NormalClass)

The collection of contiguous values associated with a counter during a sample window.

4.14.2.10 SDTextObservation (NormalClass)

The collection of text values associated with a counter during a sample window.

4.14.2.11 SDRangeRepresentation (NormalClass)

Observations of this type have a value within a certain range

4.14.2.12 SDCounterDescriptor (NormalClass)

This class is a descriptor for a counter that is constantly updated. Observations of this type are considered to be up to the moment values.

4.14.2.13 SDRepresentation (NormalClass)

This is the root of all the metadata types that describe how to render an observation. The intention of the representation class is to provide a description of observations so that a generic viewer can be provided for the observations with the same representation specialization

4.14.2.14 SDTextRepresentation (NormalClass)

Represent observations of this type as text values.

4.14.2.15 SDDiscreteRepresentation (NormalClass)

Observations of this type have discrete values (integers).

4.14.2.16 SDContiguousRepresentation (NormalClass)

Observations of this type have contiguous values (float).

4.15 *test*

4.15.1 Overview

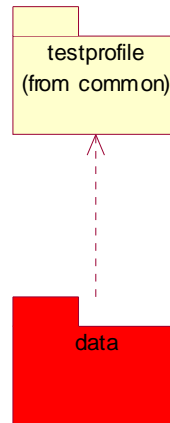


Figure 44: Package Overview

4.15.2 Classes

4.16 *data*

4.16.1 Overview

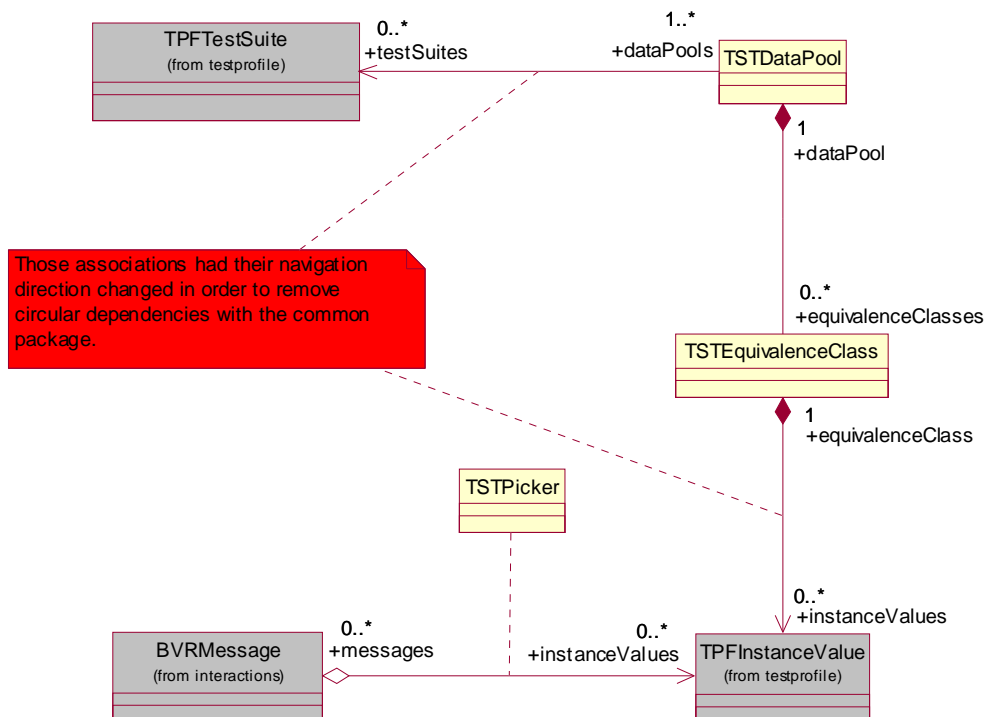


Figure 45: Main

4.16.2 Classes

4.16.2.1 TSTDataPool (NormalClass)

A DataPool is a grouping mechanism aimed at holding a set of Equivalence Classes themselves grouping a set of Instance Values.

A DataPool is referenced by TestSuite making use of it.

4.16.2.2 TSTPicker (NormalClass)

A Picker is a class realizing the association between a message and the corresponding Instance Values.

The Picker role is to decide at runtime which values need to be picked for a given message.

4.16.2.3 TSTEquivalenceClass (NormalClass)

An Equivalence class is a grouping mechanism for a set of Instance Values related. The relationship between those Instance Values is generally that their outcome is the same from the standpoint of the TestCase where they are used.

4.17 ecore

4.17.1 Overview

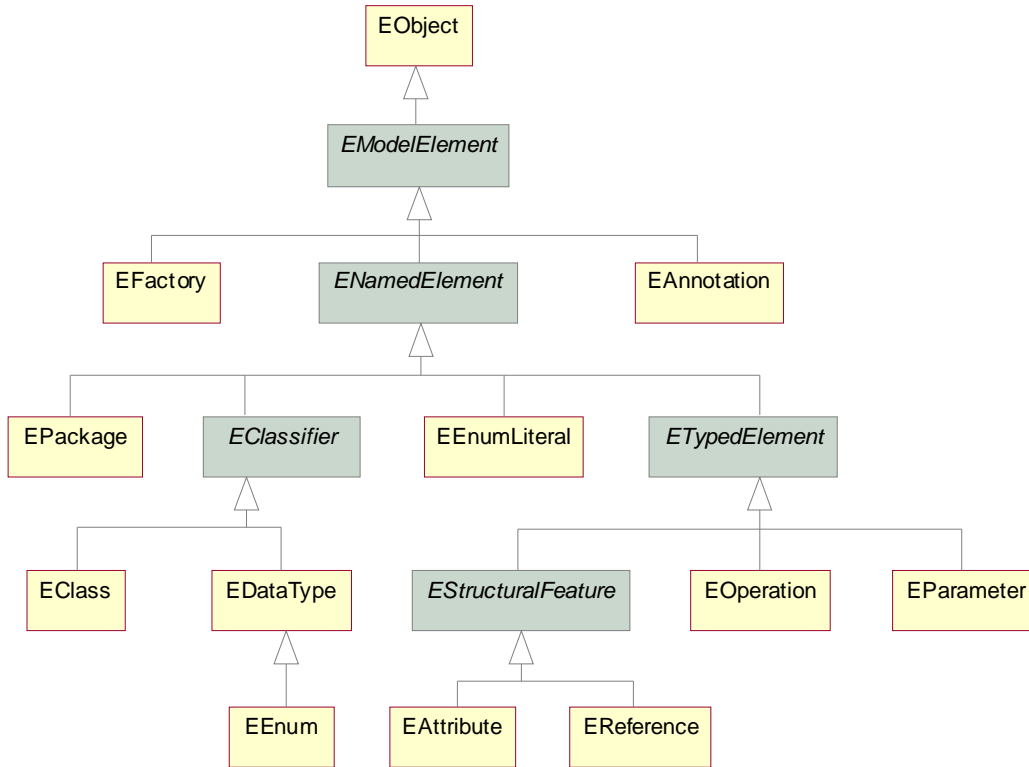


Figure 46: Ecore Hierarchy

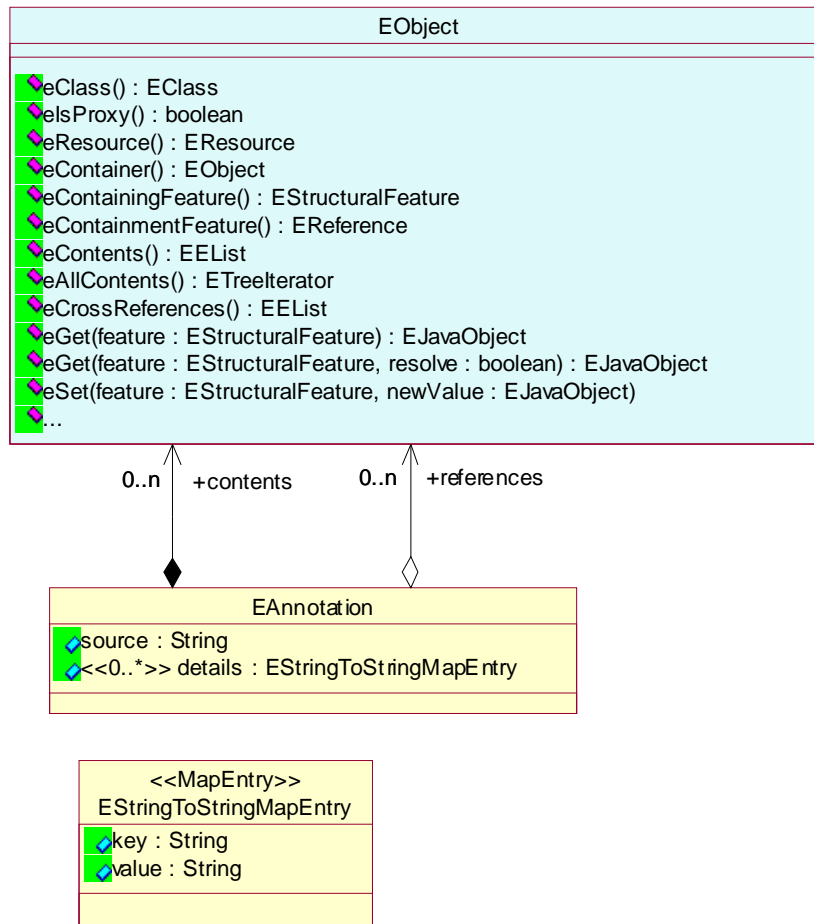


Figure 48: EObject Operations

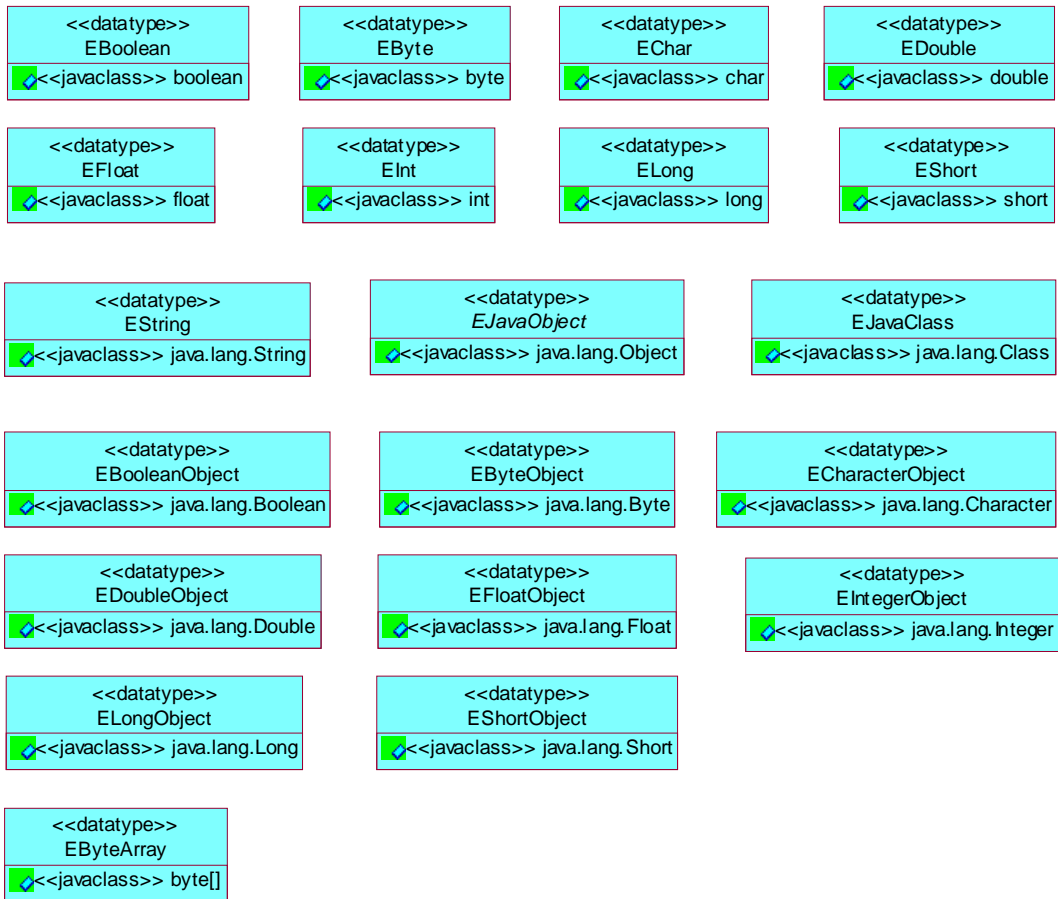


Figure 49: Java Language Types

The non-ecore types are types that are defined outside of ecore but are used as attribute types or in operation signatures by ecore classes.

Note that some types are defined as <<interface>> classes while others are defined as <<datatype>> classes.

Codegen creates java classes for the <<datatype>> classes but not for the <<interface>> classes. It is necessary to use a <<datatype>> class if that class is going to be used as the type of an attribute and if a special implementation of the createFromString or convertToString method is needed in order to serialize that attribute. For example, eJavaClass is used as the type of the instanceClass attribute of eClassifiers. If there is no need for special handling for serialization, <<interface>> classes are used because they do not require any additional classes to be generated.

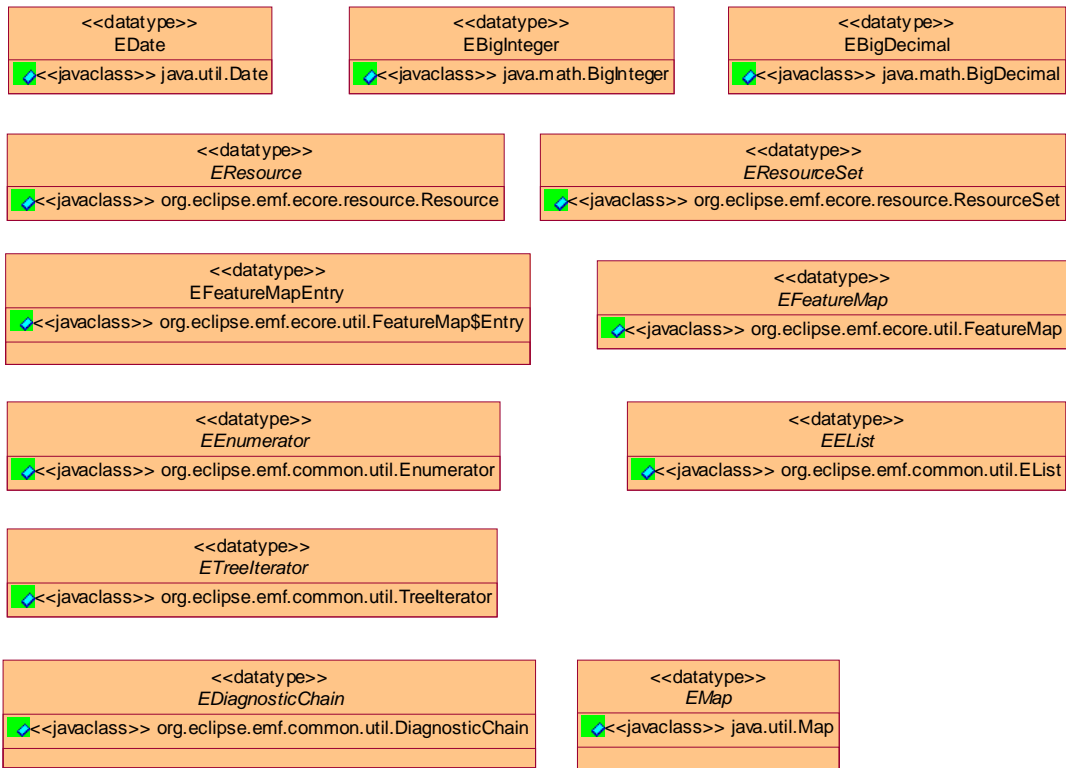


Figure 50: External Types

4.17.2 Classes

- 4.17.2.1 [EAttribute \(NormalClass\)](#)
- 4.17.2.2 [EAnnotation \(NormalClass\)](#)
- 4.17.2.3 [EClass \(NormalClass\)](#)
- 4.17.2.4 [EClassifier \(NormalClass\)](#)
- 4.17.2.5 [EDatatype \(NormalClass\)](#)
- 4.17.2.6 [EEnum \(NormalClass\)](#)
- 4.17.2.7 [EEnumLiteral \(NormalClass\)](#)
- 4.17.2.8 [EFactory \(NormalClass\)](#)
- 4.17.2.9 [EModelElement \(NormalClass\)](#)
- 4.17.2.10 [ENamedElement \(NormalClass\)](#)
- 4.17.2.11 [EObject \(NormalClass\)](#)
- 4.17.2.12 [EOperation \(NormalClass\)](#)
- 4.17.2.13 [EPackage \(NormalClass\)](#)
- 4.17.2.14 [EParameter \(NormalClass\)](#)
- 4.17.2.15 [EReference \(NormalClass\)](#)
- 4.17.2.16 [EStructuralFeature \(NormalClass\)](#)
- 4.17.2.17 [ETypedElement \(NormalClass\)](#)
- 4.17.2.18 [EBigDecimal \(NormalClass\)](#)
- 4.17.2.19 [EBigInteger \(NormalClass\)](#)
- 4.17.2.20 [EBoolean \(NormalClass\)](#)
- 4.17.2.21 [EBooleanObject \(NormalClass\)](#)

- 4.17.2.22 [EByte \(NormalClass\)](#)
- 4.17.2.23 [EByteArray \(NormalClass\)](#)
- 4.17.2.24 [EByteObject \(NormalClass\)](#)
- 4.17.2.25 [EChar \(NormalClass\)](#)
- 4.17.2.26 [ECharacterObject \(NormalClass\)](#)
- 4.17.2.27 [EDate \(NormalClass\)](#)
- 4.17.2.28 [EDiagnosticChain \(NormalClass\)](#)
- 4.17.2.29 [EDouble \(NormalClass\)](#)
- 4.17.2.30 [EDoubleObject \(NormalClass\)](#)
- 4.17.2.31 [EEList \(NormalClass\)](#)
- 4.17.2.32 [EEnumerator \(NormalClass\)](#)
- 4.17.2.33 [EFeatureMap \(NormalClass\)](#)
- 4.17.2.34 [EFeatureMapEntry \(NormalClass\)](#)
- 4.17.2.35 [EFloat \(NormalClass\)](#)
- 4.17.2.36 [EFloatObject \(NormalClass\)](#)
- 4.17.2.37 [EInt \(NormalClass\)](#)
- 4.17.2.38 [EIntegerObject \(NormalClass\)](#)
- 4.17.2.39 [EJavaClass \(NormalClass\)](#)
- 4.17.2.40 [EJavaObject \(NormalClass\)](#)
- 4.17.2.41 [ELong \(NormalClass\)](#)
- 4.17.2.42 [ELongObject \(NormalClass\)](#)
- 4.17.2.43 [EMap \(NormalClass\)](#)

- 4.17.2.44 [EResource \(NormalClass\)](#)
- 4.17.2.45 [EResourceSet \(NormalClass\)](#)
- 4.17.2.46 [EShort \(NormalClass\)](#)
- 4.17.2.47 [EShortObject \(NormalClass\)](#)
- 4.17.2.48 [EString \(NormalClass\)](#)
- 4.17.2.49 [EStringToStringMapEntry \(NormalClass\)](#)
- 4.17.2.50 [ETreeliterator \(NormalClass\)](#)

5. INTERACTION DIAGRAMS

5.1 *Top Level Interaction Diagrams*

5.2 *hierarchy*

5.3 *sdb*

5.4 *cbe*

5.5 *common*

5.6 *trace*

5.7 *statistical*

5.8 *test*

5.9 *ecore*