PPS Developer Guide

Version v0.30.0-RC2

Table of Contents

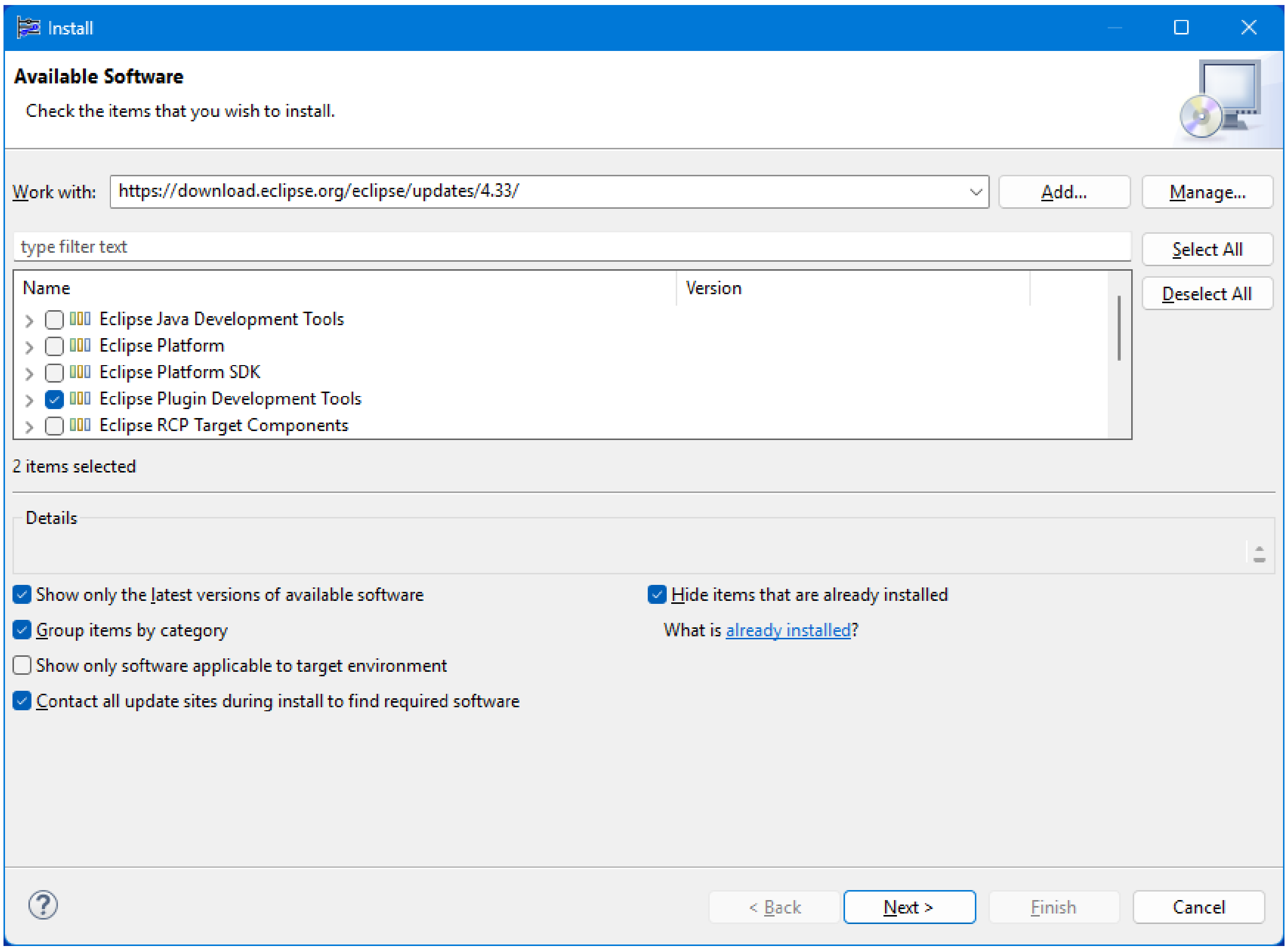
This developer guide describes how to tailor and extend the Platform Performance Suite (PPS) for a specific domain.

Reading this developer guide requires intermediate to advanced knowledge about programming for the Eclipse Modeling Framework (EMF). A good book to read as reference is EMF: Eclipse Modeling Framework.

Chapter 1. Start developing for PPS

So, you got a binary distribution of PPS and you want to tailor it for your own domain specific use? This section describes how to convert your PPS to an Eclipse PDE to customize PPS.

1. First extract the PPS distribution, choose a PPS version (i.e., windows or linux), extract it and start it, e.g. `pps.exe`.
2. Now use the **Help › Install New Software…** menu to install the `Eclipse Plugin Development Tools` from the Eclipse 2024-09 (i.e., 4.33) update site: https://download.eclipse.org/eclipse/updates/4.33/

   Available Software

   Check the items that you wish to install.

   Work with: https://download.eclipse.org/eclipse/updates/4.33/

Name	Version
☐ Eclipse Java Development Tools	
☐ Eclipse Platform	
☐ Eclipse Platform SDK	
☑ Eclipse Plugin Development Tools	
☐ Eclipse RCP Target Components	

   2 items selected

   ☑ Show only the latest versions of available software
   ☑ Hide items that are already installed
   ☑ Group items by category
   What is already installed?
   ☐ Show only software applicable to target environment
   ☑ Contact all update sites during install to find required software

3. Continue the wizard using the **[Next]** button and finally the **[Finish]** button. When asked, choose to restart the Eclipse IDE.
4. After restart install the PPS sources by using the **Help › Install New Software…** menu. Select the `PPS` category from the PPS (e.g., v0.30.0) update site: https://tno.github.io/PPS/update-site/v0.30.0/

   Please make sure to select the right update site version for your installation. You can see your installed PPS version from the **Help › About PPS vX.X** menu.

5. Continue the wizard using the **[Next]** button and finally the **[Finish]** button. When asked, choose to restart the Eclipse IDE.

Your environment is now ready for extending PPS, e.g., adding a custom TMSC reconstructor or extending its data analysis.

To start a PPS including your extensions, simply select one of your projects in the project/package explorer view and use its context menu (i.e., right mouse click) to **Run As › Eclipse Application**. A new Eclipse instance (i.e. an Eclipse runtime instance) will be executed containing all PPS features including your own features.

Chapter 2. Creating a custom TMSC reconstructor

The TMSC textual syntax allows you to create small example traces, to explore the PPS features. The textual syntax is not developed for large traces as it will suffer from scalability issues. Besides that, it is assumed that when using PPS in another domain, traces in another format are already available. In both cases it is advised to write your own *reconstructor* that converts these trace directly into a TMSC meta-model instance. This section gives an example on how to create such a custom reconstructor.

The custom TMSC reconstructor example is included in the PPS distribution. The example can be easily imported into your development environment by:

1. Use the **File › Import...** menu and then choose to import **Plug-in Development › Plug-ins and Fragments** from the tree.
2. On the next wizard page, select the `Projects with source folders` in the `Import As` section.
3. On the next wizard page filter the available plug-ins and fragments by `nl.esi.pps.tmsc.reconstruct` and click the **[Add All →]** button.
    - Now **[Finish]** the wizard to import the projects.

For more information on creating custom EMF resources or using extension points, please read the Eclipse and EMF documentation.

You can test the example by creating a `*.tmsctrace` file in a project within the Eclipse runtime instance and then use its **Reconstruct TMSC** context menu to create a TMSC for the trace.

First choose a syntax for your trace (e.g., `tmsctrace` in our example) and register it by means of the `org.eclipse.emf.ecore.extension_parser` extension point in the plugin-xml of your plugin.

Listing 1. plugin.xml of nl.esi.pps.tmsc.reconstruct

```
<?xml version="1.0" encoding="UTF-8"?>
<!--

    Copyright (c) 2018-2025 TNO and Contributors to the GitHub community

    This program and the accompanying materials are made available
    under the terms of the MIT License which is available at
    https://opensource.org/licenses/MIT

    SPDX-License-Identifier: MIT

-->
<?eclipse version="3.4"?>
<plugin>
```

```
    <extension
          point="org.eclipse.emf.ecore.extension_parser">
       <parser
             class="nl.esi.pps.tmsc.reconstruct.TmscTraceResourceFactory"
             type="tmsctrace">
       </parser>
    </extension>
 </plugin>
```

The registered `nl.esi.pps.tmsc.reconstruct.TmscTraceResourceFactory` extends `org.eclipse.emf.ecore.resource.Resource.Factory` and creates an nl.esi.pps.tmsc.reconstruct.TmscTraceResource upon request. In our example, this resource defines an interface `nl.esi.pps.tmsc.reconstruct.TmscTraceEvent` between the parser and the reconstructor to separate the concerns of parsing and model forming. The call-outs after the example provide some more information.

Listing 2. nl.esi.pps.tmsc.reconstruct.TmscTraceResource

```
/*
 * Copyright (c) 2018-2025 TNO and Contributors to the GitHub community
 *
 * This program and the accompanying materials are made available
 * under the terms of the MIT License which is available at
 * https://opensource.org/licenses/MIT
 *
 * SPDX-License-Identifier: MIT
 */
package nl.esi.pps.tmsc.reconstruct;

import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.LineNumberReader;
import java.util.Map;

import org.eclipse.emf.common.util.URI;
import org.eclipse.emf.ecore.resource.impl.ResourceImpl;

public class TmscTraceResource extends ResourceImpl { ①
    public TmscTraceResource() {
        super();
    }

    public TmscTraceResource(URI uri) {
        super(uri);
    }

    @Override
    protected void doLoad(InputStream inputStream, Map<?, ?> options) throws
IOException {
        TmscTraceReconstructor reconstructor = new TmscTraceReconstructor(); ②
        reconstructor.preReconstruct();

        try (LineNumberReader reader = new LineNumberReader(new InputStreamReader
(inputStream))) {
            try {
                String line;
                while ((line = reader.readLine()) != null) {
                    TmscTraceEvent traceEvent = TmscTraceParser.parseLine(line); ③
                    reconstructor.reconstruct(traceEvent);
                }
            } catch (RuntimeException e) {
                throw new IOException("Failed to parse trace at line " + reader
.getLineNumber(), e);
            }
        }

        reconstructor.postReconstruct(); ④
        getContents().add(reconstructor.getTmsc());
        getContents().add(reconstructor.getArchitecture());
    }
}
```

① Extending `org.eclipse.emf.ecore.resource.impl.ResourceImpl` allows you to implement the `doLoad(InputStrweam, Map)` method. If you also need to be able to save a TMSC model instance to your trace format, please implement the `doSave(OutputStream, Map)` method.

② At the beginning of the load, the `TmscReconstructor` is initialized by calling its `preReconstruct()` method. This will prepare the Contents of architecture and Contents of tmsc.

③ For every line in the trace, a `TmscTraceEvent` is created by the `TmscTraceParser` and this event is fed to the `TmscReconstructor` by calling its `reconstruct(TmscTraceEvent)` method.

④ The TMSC model instance is *well-formed* by calling the `postReconstruct()` method on the `TmscReconstructor`. As part of the postReconstruct, some generic `nl.esi.pps.tmsc.util.TmscRefinements` are called, please see the `TmscReconstructor` implementation. Finally, the reconstructed TMSC and architecture are added to the resource contents.

Chapter 3. Extending Data Analysis

The Data Analysis view for the TMSC graphical viewer shows either a time series or a histogram of the *'duration'* for all *'siblings'* of the selected element. The *'duration'* of an element can be any time-related aspect of an element, e.g. its absolute (observed) duration or its time-bound. The *'siblings'* of an element are all occurrences in the model for which their *'duration'* is comparable, in other words, it makes sense to compare them. This already sounds very abstract and besides that, it is not known which elements are added to the TMSC model by means of extensions (e.g. analysis or customer specific model extensions). For this reason an extensible approach is designed to provide the input for data analysis.

This design is implemented by means of reusing the EMF Edit ItemProvider adapter factories extension point in combination with our own `nl.esi.pps.tmsc.edit.dataAnalysis` extension point. EMF adapters are a very powerful technique and EMF Edit's ItemProviders are designed to automatically add UI adapters to an EMF model instance. For more information on adapters and adapter factories, see https://eclipsesource.com/blogs/2013/01/29/emf-itemprovider-magic/ or the book EMF: Eclipse Modeling Framework.

The `nl.esi.pps.tmsc.edit.dataAnalysis` extension point adds a flexible approach on top, that allows to register an nl.esi.pps.tmsc.provider.dataanalysis.IDataAnalysisItemContentProvider per `EClass`. This item provider needs to be registered to the EMF framework in order to be found by the data analysis view. For this, an `nl.esi.pps.tmsc.provider.dataanalysis.internal.DataAnalysisItemContentProvider` is registered as EMF Edit ItemProvider via the `nl.esi.pps.tmsc.provider.dataanalysis.DataAnalysisItemContentProviderAdapterFactory`, see the plugin.xml of nl.esi.pps.tmsc.edit. We only register the `nl.esi.pps.tmsc.provider.dataanalysis.IDataAnalysisItemContentProvider` as `supportedTypes` and implement the default create method to returning an instance of the `nl.esi.pps.tmsc.provider.dataanalysis.internal.DataAnalysisItemContentProvider`. We can use a single instance as our implementation is stateless. This instance is disposed when the adapter factory is disposed.

Registering an EMF Edit ItemProvider only needs to be done once per language that wants to support data analysis!

The nl.esi.pps.tmsc.provider.dataanalysis.IDataAnalysisItemContentProvider interface defines the API to populate the data analysis views. The API is explained in the JavaDoc of the interface. All methods take an `Object` as argument, this is the model element for which the data is requested. First the `Set<String> getConfigurations(Object)` will be invoked. This method should return all supported data analysis configurations for the model element. An example of supporting different configurations is implemented in `nl.esi.pps.tmsc.provider.ext.ExecutionExtItemProvider` where *'siblings'* can either be executions on the same `Lifeline` or executions executed by the same `Component`. The configurations will appear as tabs in the data analysis view. In all other methods, both the model element as the user selected configuration at least will be provided.

Within Eclipse, use `Ctrl` + `Shift` + `T` to open a type based on its name.

Listing 3. nl.esi.pps.tmsc.provider.dataanalysis.IDataAnalysisItemContentProvider

```
/**
 * This interface defines the required information to populate the Data Analysis
 * view of the TMSC editor.
 */
public interface IDataAnalysisItemContentProvider {
    /**
     * The default configuration for data analysis. If your data analysis only
     * supports a single configuration, its is recommended to use this
     * configuration.
     */
    static final String DEFAULT_CONFIGURATION = "Default";

    /**
     * Returns the supported data analysis configurations for {@code object}.
     */
    Set<String> getConfigurations(Object object);

    /**
     * Returns the title of the data analysis for {@code object} in the selected
     * {@code configuration}.
     */
    String getTitle(Object object, String configuration);

    /**
     * Returns the budget of the data analysis for {@code object} in the selected
     * {@code configuration}, or {@code null} if the budget is not applicable.
     */
    default Long getBudget(Object object, String configuration) {
        return null;
    }

    /**
     * Returns the siblings of {@code object} for the selected
     * {@code configuration}.
     */
    Iterable<?> getSiblings(Object object, String configuration);

    /**
     * Returns the duration of {@code sibling} in the selected
     * {@code configuration}. Please note that {@code sibling} can be any of the
     * elements as returned by {@link #getSiblings(Object, String)} or the
     * {@code object} itself.
     */
    Long getDuration(Object object, Object sibling, String configuration);

    /**
     * @return (@code true} if the bars in the time-series viewer should be colored
     *         based on a {@link #getCategory(Object, Object, String) category}.
     */
    default boolean isCategorized(Object object, String configuration) {
        return false;
    }

    /**
     * @return the category for the {@code sibling} in the time-series viewer.
     */
    default String getCategory(Object object, Object sibling, String configuration) {
        return DEFAULT_CONFIGURATION;
    }
}
```

The only thing left to do is to register the data analysis item content provider to the EMF framework, which is done by means of the `nl.esi.pps.tmsc.edit.dataAnalysis` extension point in the plugin.xml.

Listing 4. plugin.xml of nl.esi.pps.tmsc.edit

```
   <extension point="org.eclipse.emf.edit.itemProviderAdapterFactories"> ①
      <factory
            uri="http://www.esi.nl/pps/tmsc"
            class=
"nl.esi.pps.tmsc.provider.dataanalysis.DataAnalysisItemContentProviderAdapterFactory"
            supportedTypes=
"nl.esi.pps.tmsc.provider.dataanalysis.IDataAnalysisItemContentProvider"/>
   </extension>
   <extension point="nl.esi.pps.tmsc.edit.dataAnalysis"> ②
      <DataAnalysis
            content_provider=
"nl.esi.pps.tmsc.provider.dataanalysis.ExecutionDataAnalysisItemContentProvider"
            eclass_uri="http://www.esi.nl/pps/tmsc#Execution"
            id=
"nl.esi.pps.tmsc.provider.dataanalysis.ExecutionDataAnalysisItemContentProvider">
      </DataAnalysis>
   </extension>
```

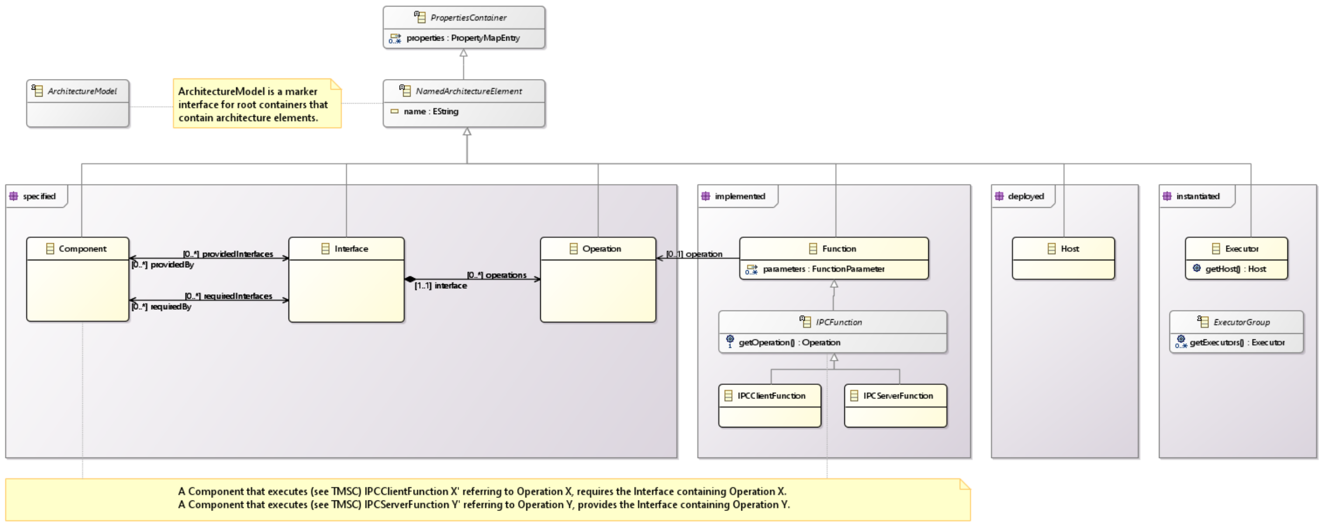
① An additional EMF-edit item provider adapter factory is registered for the `http://www.esi.nl/pps/tmsc` language to bind the `nl.esi.pps.tmsc.edit.dataAnalysis` extension point, see the `nl.esi.pps.tmsc.provider.dataanalysis.DataAnalysisItemContentProviderAdapterFactory` code for more information.

② The `nl.esi.pps.tmsc.provider.dataanalysis.ExecutionDataAnalysisItemContentProvider` is registered for the `http://www.esi.nl/pps/tmsc#//Execution` EClass

Chapter 4. PPS meta-models

4.1. Contents of architecture

The architecture meta-model defines the minimal set of concepts that are required to define the TMSC meta-model. The architecture package contains sub-packages that represent the lifecycle stages of an architecture as inspired by Koziolek et al.[1], where the author presents a view on the software component lifecycle. This study considers 4 levels of lifecycle stages: the specification, implementation, deployment and runtime level. This architecture considers an additional distinction in the runtime level, which is divided in the (runtime) instantiated and (runtime) stimulated stage. The TMSC meta-model defines concepts that belong to the stimulated stage.



ArchitectureModel is a marker interface for root containers that contain architecture elements.

A Component that executes (see TMSC) IPCClientFunction X' referring to Operation X, requires the Interface containing Operation X.
A Component that executes (see TMSC) IPCServerFunction Y' referring to Operation Y, provides the Interface containing Operation Y.

Figure 1. Diagram of the architecture meta-model

Ns Prefix

architecture

Ns URI

http://www.esi.nl/pps/architecture

4.1.1. Types

4.1.1.1. Interface ArchitectureModel

ArchitectureModel is a marker interface for root containers that contain architecture elements. Is is not required for an architecture to implement the interface, but it may speed up certain analyses.

Used at

- `tmsc.FullScopeTMSC.architectures`

4.1.1.2. Abstract Class NamedArchitectureElement

Super-types

- `properties.PropertiesContainer`

Sub-types

- `deployed.Host`
- `implemented.Function`
- `implemented.FunctionParameter`
- `implemented.IPCClientFunction`
- `implemented.IPCFunction`
- `implemented.IPCServerFunction`
- `instantiated.Executor`
- `specified.Component`
- `specified.Interface`
- `specified.Operation`

Attributes

Name	Type	Properties	Description
`name`	`EString`		

4.2. Contents of deployed

In the Deployed lifecycle stage, the implemented software components are built and bundled into binaries. At this stage, the binaries are placed into a filesystem from where they can be installed to the different hosts. The installation takes place during the transition from Deployed to Instantiated stage. During this transition the binaries are mapped (installed) to specific hosts. We consider for the sake of simplicity that the deployment of binaries onto hosts is static and performed before the system is initiated/started.

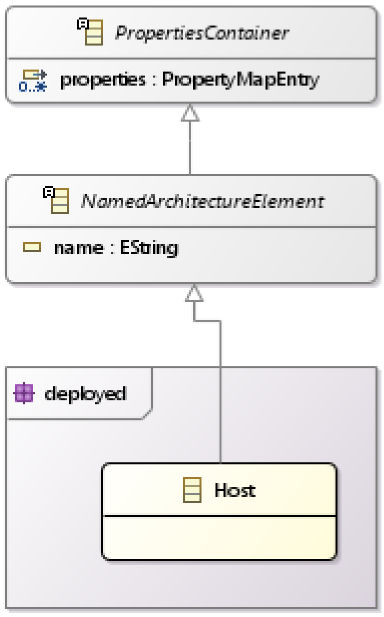


Figure 2. Diagram of the deployed lyfecycle stage in the architecture meta-model

Ns Prefix

architecture_deployed

Ns URI

http://www.esi.nl/pps/architecture/deployed

4.2.1. Types

4.2.1.1. Class Host

Super-types

- `architecture.NamedArchitectureElement`
- `properties.PropertiesContainer`

Used at

- `instantiated.Executor.getHost()`

4.3. Contents of implemented

In the Implementation lifecycle stage, the specified software components and interfaces are implemented. Depending on the software architecture, as well as the execution platform, a component may have more than one implementation, which may be related to a specific execution platform (i.e., programming language, type of processing unit, etc.) or other factors (i.e., quality of service, etc.)

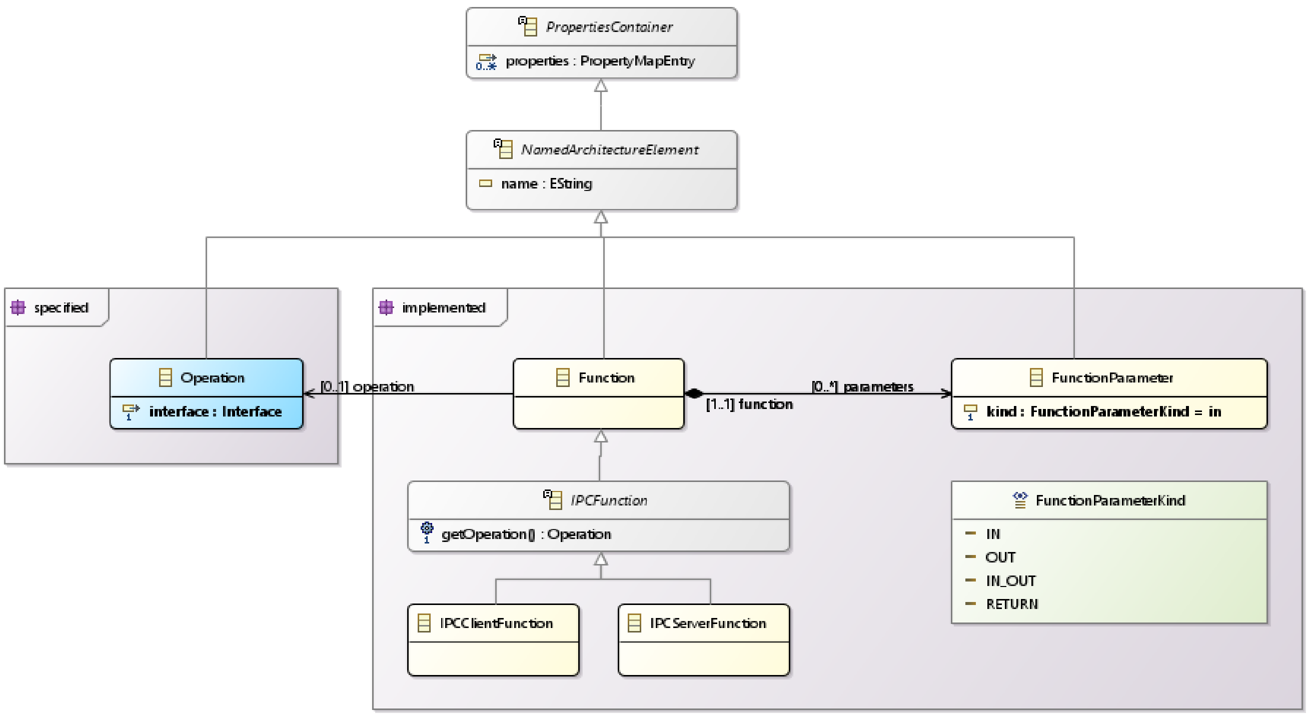


Figure 3. Diagram of the implemented lyfecycle stage in the architecture meta-model

Ns Prefix

architecture_implemented

Ns URI

http://www.esi.nl/pps/architecture/implemented

4.3.1. Enumerations

4.3.1.1. FunctionParameterKind

Default Value

`in`

Literals

Name	Value	Literal	Description
`IN`	0	`in`	
`OUT`	1	`out`	
`IN_OUT`	2	`inout`	
`RETURN`	3	`return`	

Used at

- `implemented.FunctionParameter.kind`

4.3.2. Types

4.3.2.1. Class Function

Super-types

- `architecture.NamedArchitectureElement`
- `properties.PropertiesContainer`

Sub-types

- `implemented.IPCClientFunction`
- `implemented.IPCFunction`
- `implemented.IPCServerFunction`

Containments

Name	Type	Properties	Description
`parameters`	`implemented.FunctionParameter` *EOpposite:* `function`		

References

Name	Type	Properties	Description
operation ▯ implemented. IPCFunction. getOperation()	specified. Operation		

Used at

- implemented.FunctionParameter.function
- tmsc.EntryEvent.function
- tmsc.Event.function
- tmsc.Execution.function
- tmsc.ExitEvent.function

4.3.2.2. Class FunctionParameter

Super-types

- architecture.NamedArchitectureElement
- properties.PropertiesContainer

Attributes

Name	Type	Properties	Description
kind	implemented. FunctionParamete rKind	[1]	

References

Name	Type	Properties	Description
function	implemented. Function *EOpposite:* parameters	non- resolveProxies **container** [1]	

Used at

- implemented.Function.parameters
- implemented.IPCClientFunction.parameters
- implemented.IPCFunction.parameters
- implemented.IPCServerFunction.parameters
- tmsc.FunctionArgumentMapEntry.key

4.3.2.3. Class IPCClientFunction

A Component that executes (see TMSC) IPCClientFunction X' referring to Operation X, requires the

Interface containing Operation X.

Super-types

- `architecture.NamedArchitectureElement`
- `implemented.Function`
- `implemented.IPCFunction`
- `properties.PropertiesContainer`

4.3.2.4. Abstract Class IPCFunction

Super-types

- `architecture.NamedArchitectureElement`
- `implemented.Function`
- `properties.PropertiesContainer`

Sub-types

- `implemented.IPCClientFunction`
- `implemented.IPCServerFunction`

Operations

Name	Aspect and Type	Properties	Description
abstract `getOperation()` ▯ `implemented.Function.operation`	*returns* `specified.Operation`	`[1]`	

4.3.2.5. Class IPCServerFunction

A Component that executes (see TMSC) IPCServerFunction Y' referring to Operation Y, provides the Interface containing Operation Y.

Super-types

- `architecture.NamedArchitectureElement`
- `implemented.Function`
- `implemented.IPCFunction`
- `properties.PropertiesContainer`

4.4. Contents of instantiated

At instantiated stage the system is started, and the components are instantiated and start living in memory. Here, the runtime component takes form and is mapped on OS execution entities, i.e., on a specific thread which is mapped on specific process, etc. Communication ports are created and instantiated enabling communication among the runtime components. The OS execution entities, i.e., processes and threads are mapped on specific processing unit, with specific scheduling policies, priorities, etc.

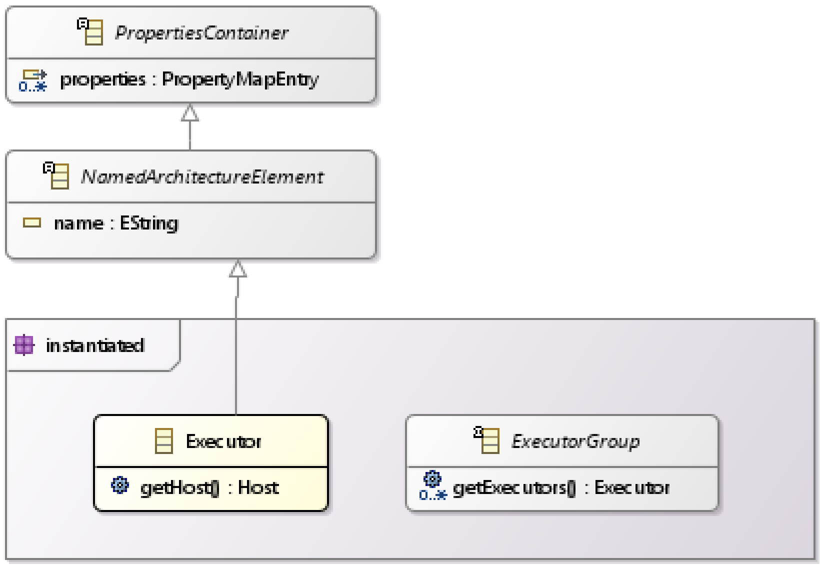


Figure 4. Diagram of the instantiated lyfecycle stage in the architecture meta-model

Ns Prefix

architecture_instantiated

Ns URI

http://www.esi.nl/pps/architecture/instantiated

4.4.1. Types

4.4.1.1. Class Executor

Super-types

- `architecture.NamedArchitectureElement`
- `properties.PropertiesContainer`

Operations

Name	Aspect and Type	Properties	Description
abstract `getHost()`	*returns* `deployed.Host`		Typically an executor is deployed onto a host, but PPS doesn't make any assumptions on the kind of relation, hence this relation is merely implemented as an operation and it is up to the architecture implementer to define the relation. The default implementation of this method returns the container of type Host in the hierarchy of this Executor.

Used at

- `instantiated.ExecutorGroup.getExecutors()`
- `tmsc.Lifeline.executor`

4.4.1.2. Interface ExecutorGroup

The ExecutorGroup allows to create logical groups of Executors that have a relation. When selecting an ExecutorGroup in the Outline view of the TMSC viewer, all its Executors will be visualized.

Operations

Name	Aspect and Type	Properties	Description
abstract `getExecutors()`	*returns* `instantiated.Executor`	unique	Typically an ExecutorGroup contains Executors, but PPS doesn't make any assumptions on the kind of relation, hence this relation is merely implemented as an operation and it is up to the architecture implementer to define the relation. The default implementation of this method returns all first-level descendants of this ExecutorGroup that are of kind Executor, meaning if an Executor contains another Executor, the latter will not be included in the result.

4.5. Contents of metric

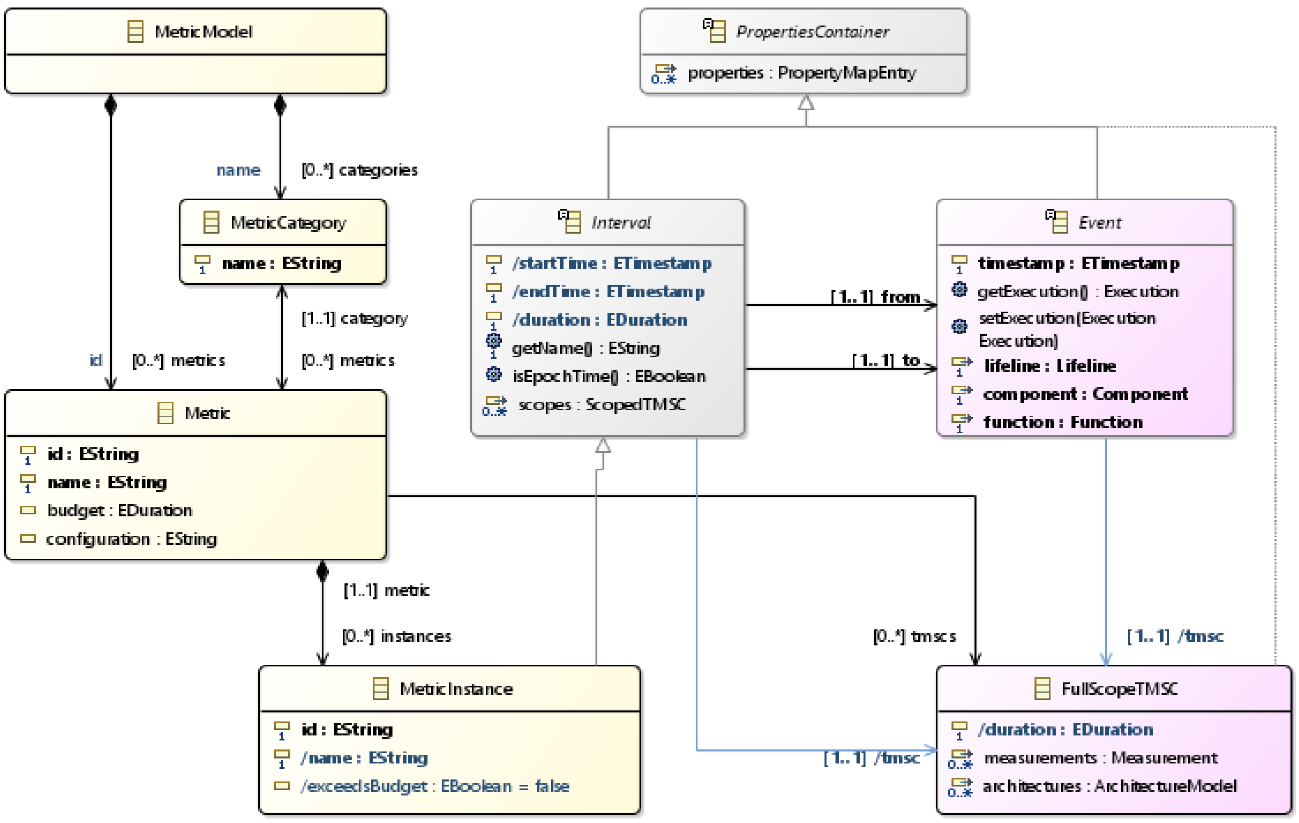


Figure 5. Diagram of the Metric meta-model

Ns Prefix

tmsc_metric

Ns URI

http://www.esi.nl/tmsc/metric

4.5.1. Types

4.5.1.1. Class Metric

Attributes

Name	Type	Properties	Description
budget	tmsc.EDuration		
configuration	EString		
id	EString	[1]	
name	EString	[1]	

Containments

Name	Type	Properties	Description
`instances`	`metric. MetricInstance` *EOpposite:* `metric`	*EKeys:* `id`	

References

Name	Type	Properties	Description
`category`	`metric. MetricCategory` *EOpposite:* `metrics`	`[1]`	
`tmscs`	`tmsc. FullScopeTMSC`		The TMSCs that were analyzed to resolve the Metric instances that are contained by this Metric.

Used at

- `metric.MetricCategory.metrics`
- `metric.MetricInstance.metric`
- `metric.MetricModel.metrics`

4.5.1.2. Class MetricCategory

Attributes

Name	Type	Properties	Description
`name`	`EString`	`[1]`	

References

Name	Type	Properties	Description
`metrics`	`metric.Metric` *EOpposite:* `category`		

Used at

- `metric.Metric.category`
- `metric.MetricModel.categories`

4.5.1.3. Class MetricInstance

Super-types

- `properties.PropertiesContainer`

- tmsc.ITimeRange
- tmsc.Interval

Attributes

Name	Type	Properties	Description
exceedsBudget	EBoolean	unchangeable derived transient volatile	
id	EString	[1]	
name tmsc.Interval.getName()	EString	[1] unchangeable derived transient volatile	

References

Name	Type	Properties	Description
metric	metric.Metric *EOpposite:* instances	**container** [1]	

Used at

- metric.Metric.instances

4.5.1.4. Class MetricModel

Containments

Name	Type	Properties	Description
categories	metric. MetricCategory	*EKeys:* name	
metrics	metric.Metric	*EKeys:* id	

4.6. Contents of properties

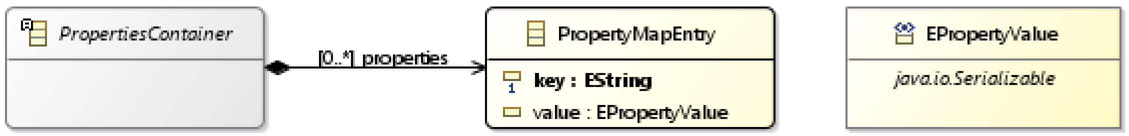


Figure 6. Diagram of the properties meta-model

Ns Prefix

properties

Ns URI

http://www.esi.nl/properties

4.6.1. Data Types

4.6.1.1. EPropertyValue

Instance Type Name

`java.io.Serializable`

Used at

- `properties.PropertyMapEntry.value`

4.6.2. Types

4.6.2.1. Abstract Class PropertiesContainer

The PropertiesContainer facilitates the seamless extension of specific modeling elements with additional attributes, that are required for specific type of analysis. Though this allows rapid prototyping or customer specific annotations, for every property that is added to the model the question should be raised if this property is a primary citizen of the meta-model. If so, the property should be promoted as attribute or reference in the meta-model.

Sub-types

- `architecture.NamedArchitectureElement`
- `deployed.Host`
- `implemented.Function`
- `implemented.FunctionParameter`
- `implemented.IPCClientFunction`
- `implemented.IPCFunction`
- `implemented.IPCServerFunction`
- `instantiated.Executor`
- `metric.MetricInstance`
- `specified.Component`
- `specified.Interface`
- `specified.Operation`
- `tmsc.Dependency`
- `tmsc.DomainDependency`
- `tmsc.EntryEvent`
- `tmsc.Event`
- `tmsc.Execution`

- tmsc.ExitEvent
- tmsc.FullScopeTMSC
- tmsc.Interval
- tmsc.Lifeline
- tmsc.LifelineSegment
- tmsc.Measurement
- tmsc.Message
- tmsc.MessageControl
- tmsc.Reply
- tmsc.Request
- tmsc.ScopedTMSC
- tmsc.TMSC

Containments

Name	Type	Properties	Description
properties	properties.PropertyMapEntry		

4.6.2.2. Class PropertyMapEntry

Instance Type Name

`java.util.Map$Entry`

Attributes

Name	Type	Properties	Description
key	EString	[1]	
value	properties.EPropertyValue		

Used at

- architecture.NamedArchitectureElement.properties
- deployed.Host.properties
- implemented.Function.properties
- implemented.FunctionParameter.properties
- implemented.IPCClientFunction.properties
- implemented.IPCFunction.properties
- implemented.IPCServerFunction.properties
- instantiated.Executor.properties
- metric.MetricInstance.properties

- `properties.PropertiesContainer.properties`
- `specified.Component.properties`
- `specified.Interface.properties`
- `specified.Operation.properties`
- `tmsc.Dependency.properties`
- `tmsc.DomainDependency.properties`
- `tmsc.EntryEvent.properties`
- `tmsc.Event.properties`
- `tmsc.Execution.properties`
- `tmsc.ExitEvent.properties`
- `tmsc.FullScopeTMSC.properties`
- `tmsc.Interval.properties`
- `tmsc.Lifeline.properties`
- `tmsc.LifelineSegment.properties`
- `tmsc.Measurement.properties`
- `tmsc.Message.properties`
- `tmsc.MessageControl.properties`
- `tmsc.Reply.properties`
- `tmsc.Request.properties`
- `tmsc.ScopedTMSC.properties`
- `tmsc.TMSC.properties`

4.7. Contents of specified

In this lifecycle stage, the specification of the software components takes place. Provided and required interfaces describe the contracts between the software components and encode the service specification. At this stage, the software interfaces are also specified by bundling operations that each of those implements.

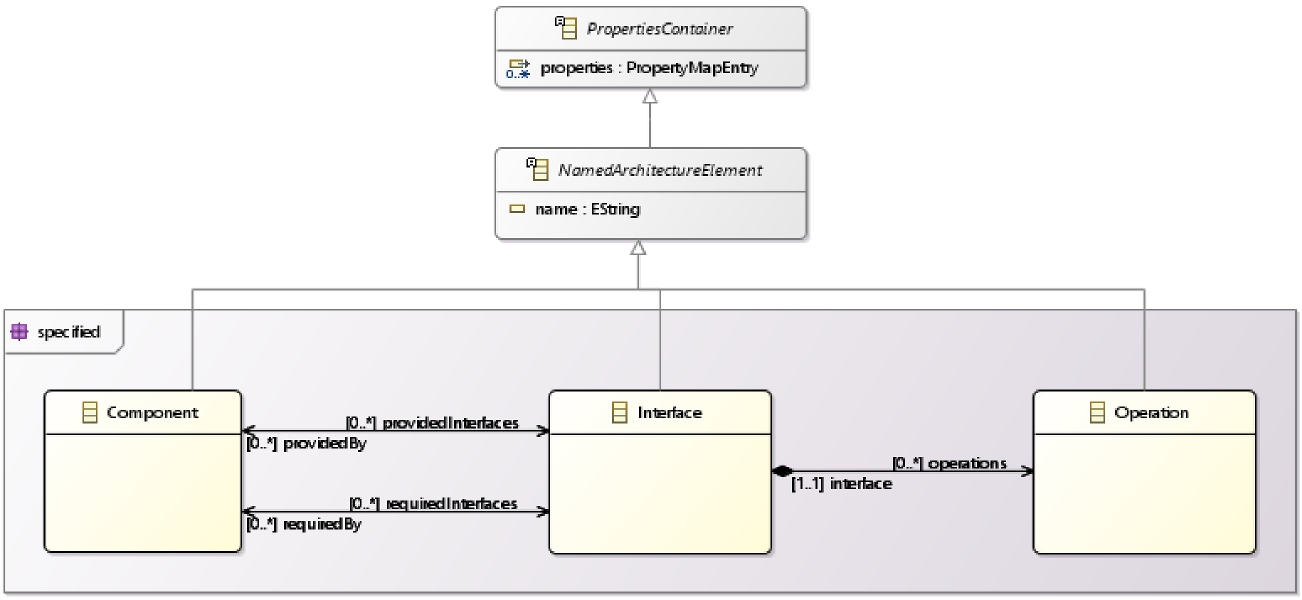


Figure 7. Diagram of the specified lyfecycle stage in the architecture meta-model

Ns Prefix

architecture_specified

Ns URI

http://www.esi.nl/pps/architecture/specified

4.7.1. Types

4.7.1.1. Class Component

Super-types

- `architecture.NamedArchitectureElement`
- `properties.PropertiesContainer`

References

Name	Type	Properties	Description
`providedInterfaces`	`specified.Interface` *EOpposite:* `providedBy`		
`requiredInterfaces`	`specified.Interface` *EOpposite:* `requiredBy`		

Used at

- `specified.Interface.providedBy`
- `specified.Interface.requiredBy`

- tmsc.EntryEvent.component
- tmsc.Event.component
- tmsc.Execution.component
- tmsc.ExitEvent.component

4.7.1.2. Class Interface

Super-types

- architecture.NamedArchitectureElement
- properties.PropertiesContainer

Containments

Name	Type	Properties	Description
operations	specified.Operation *EOpposite:* interface		

References

Name	Type	Properties	Description
providedBy	specified.Component *EOpposite:* providedInterfaces		
requiredBy	specified.Component *EOpposite:* requiredInterfaces		

Used at

- specified.Component.providedInterfaces
- specified.Component.requiredInterfaces
- specified.Operation.interface

4.7.1.3. Class Operation

Super-types

- architecture.NamedArchitectureElement
- properties.PropertiesContainer

References

Name	Type	Properties	Description
interface	specified.Interface *EOpposite:* operations	**container** [1]	

Used at

- implemented.Function.operation
- implemented.IPCClientFunction.operation
- implemented.IPCFunction.getOperation()
- implemented.IPCFunction.getOperation()
- implemented.IPCServerFunction.operation
- implemented.IPCFunction.getOperation()
- specified.Interface.operations

4.8. Contents of tmsc

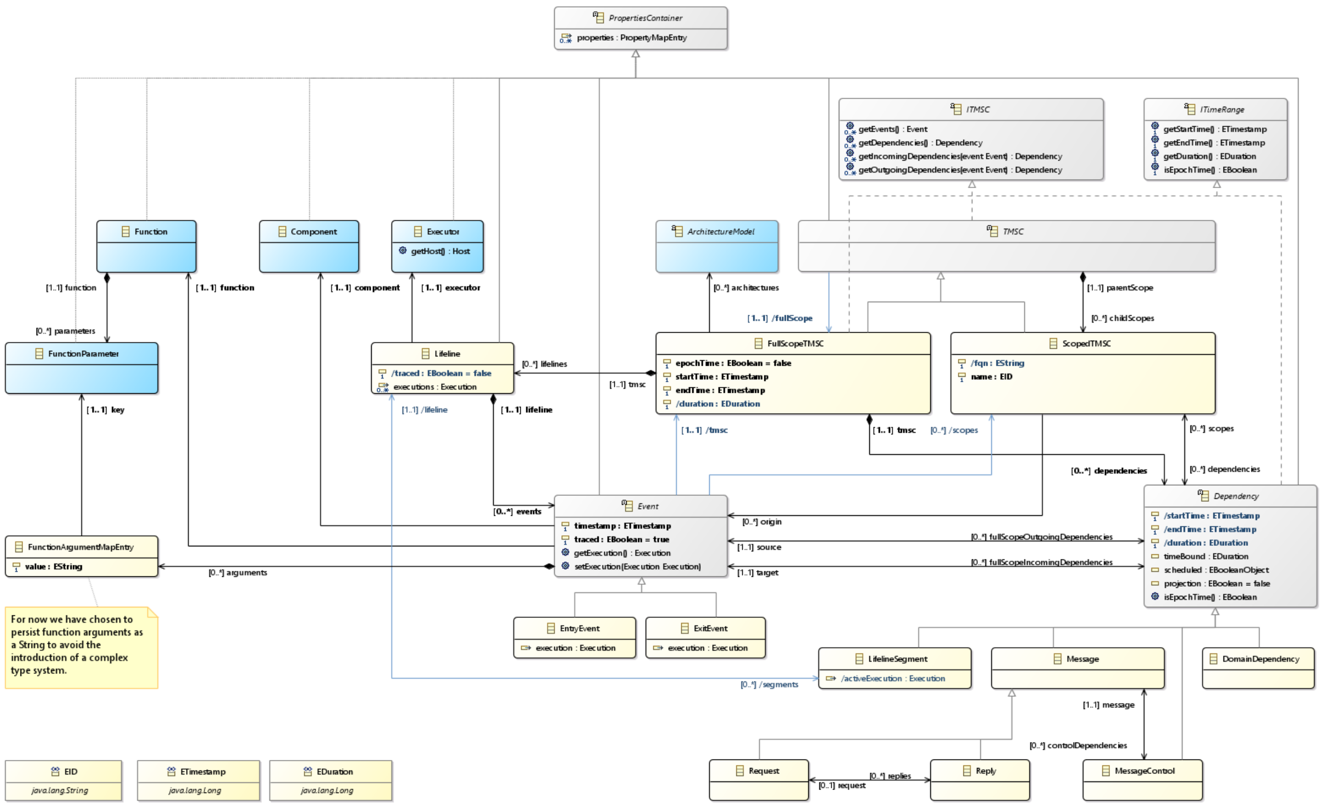


Figure 8. Diagram of the TMSC meta-model

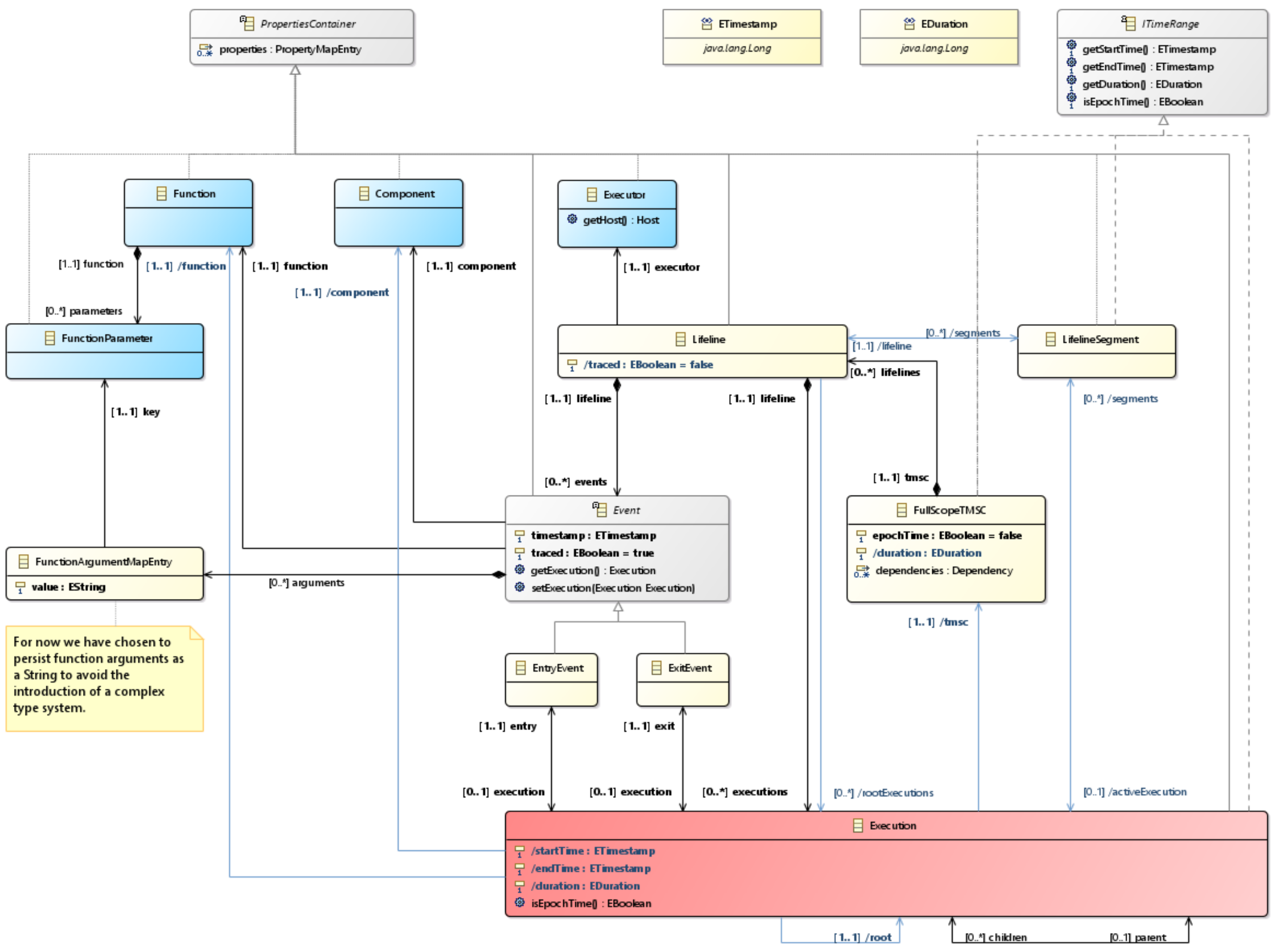


Figure 9. Diagram of call-stacks in the TMSC meta-model

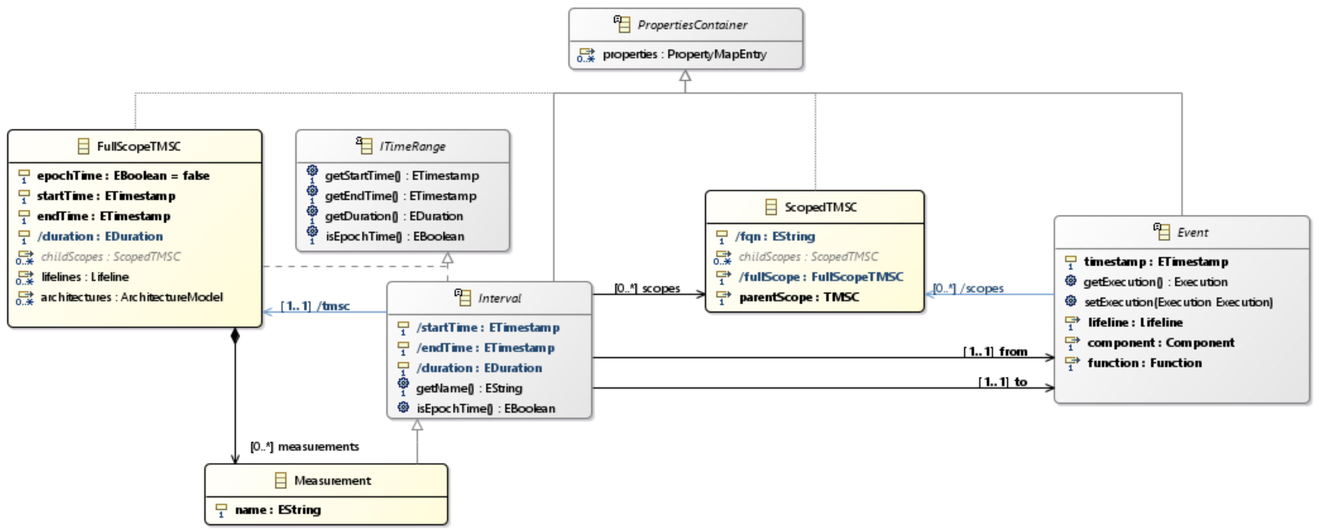


Figure 10. Diagram of measurments in the TMSC meta-model

Ns Prefix

tmsc

Ns URI

http://www.esi.nl/pps/tmsc

4.8.1. Data Types

4.8.1.1. EDuration

Represents a duration in nanoseconds.

Instance Type Name

`java.lang.Long`

Used at

- `metric.Metric.budget`
- `metric.MetricInstance.duration`
- `tmsc.ITimeRange.getDuration()`
- `tmsc.Dependency.duration`
- `tmsc.Dependency.timeBound`
- `tmsc.DomainDependency.duration`
- `tmsc.DomainDependency.timeBound`
- `tmsc.ITimeRange.getDuration()`
- `tmsc.Execution.duration`
- `tmsc.FullScopeTMSC.duration`
- `tmsc.ITimeRange.getDuration()`
- `tmsc.Interval.duration`
- `tmsc.LifelineSegment.duration`
- `tmsc.LifelineSegment.timeBound`
- `tmsc.ITimeRange.getDuration()`
- `tmsc.Measurement.duration`
- `tmsc.ITimeRange.getDuration()`
- `tmsc.Message.duration`
- `tmsc.Message.timeBound`
- `tmsc.ITimeRange.getDuration()`
- `tmsc.MessageControl.duration`
- `tmsc.MessageControl.timeBound`
- `tmsc.ITimeRange.getDuration()`
- `tmsc.Reply.duration`
- `tmsc.Reply.timeBound`
- `tmsc.ITimeRange.getDuration()`
- `tmsc.Request.duration`
- `tmsc.Request.timeBound`

- `tmsc.ITimeRange.getDuration()`

4.8.1.2. EID

Instance Type Name

`java.lang.String`

Used at

- `tmsc.ScopedTMSC.name`

4.8.1.3. ETimestamp

Represents a time stamp in nanoseconds. The timestamp can be either relative (i.e. relative to 0) or absolute (i.e. relative to the epoch). Please note that {@link FullScopeTMSC#setEpochTime(boolean)} should be set accordingly.

Instance Type Name

`java.lang.Long`

Used at

- `metric.MetricInstance.endTime`
- `metric.MetricInstance.startTime`
- `tmsc.ITimeRange.getEndTime()`
- `tmsc.ITimeRange.getStartTime()`
- `tmsc.Dependency.endTime`
- `tmsc.Dependency.startTime`
- `tmsc.DomainDependency.endTime`
- `tmsc.DomainDependency.startTime`
- `tmsc.ITimeRange.getEndTime()`
- `tmsc.ITimeRange.getStartTime()`
- `tmsc.EntryEvent.timestamp`
- `tmsc.Event.timestamp`
- `tmsc.Execution.endTime`
- `tmsc.Execution.startTime`
- `tmsc.ExitEvent.timestamp`
- `tmsc.FullScopeTMSC.endTime`
- `tmsc.FullScopeTMSC.startTime`
- `tmsc.ITimeRange.getEndTime()`
- `tmsc.ITimeRange.getStartTime()`
- `tmsc.Interval.endTime`
- `tmsc.Interval.startTime`

- `tmsc.LifelineSegment.endTime`
- `tmsc.LifelineSegment.startTime`
- `tmsc.ITimeRange.getEndTime()`
- `tmsc.ITimeRange.getStartTime()`
- `tmsc.Measurement.endTime`
- `tmsc.Measurement.startTime`
- `tmsc.ITimeRange.getEndTime()`
- `tmsc.ITimeRange.getStartTime()`
- `tmsc.Message.endTime`
- `tmsc.Message.startTime`
- `tmsc.ITimeRange.getEndTime()`
- `tmsc.ITimeRange.getStartTime()`
- `tmsc.MessageControl.endTime`
- `tmsc.MessageControl.startTime`
- `tmsc.ITimeRange.getEndTime()`
- `tmsc.ITimeRange.getStartTime()`
- `tmsc.Reply.endTime`
- `tmsc.Reply.startTime`
- `tmsc.ITimeRange.getEndTime()`
- `tmsc.ITimeRange.getStartTime()`
- `tmsc.Request.endTime`
- `tmsc.Request.startTime`
- `tmsc.ITimeRange.getEndTime()`
- `tmsc.ITimeRange.getStartTime()`

4.8.2. Types

4.8.2.1. Abstract Class Dependency

Super-types

- `properties.PropertiesContainer`
- `tmsc.ITimeRange`

Sub-types

- `tmsc.DomainDependency`
- `tmsc.LifelineSegment`
- `tmsc.Message`

- tmsc.MessageControl
- tmsc.Reply
- tmsc.Request

Attributes

Name	Type	Properties	Description
duration tmsc.ITimeRange. getDuration()	tmsc.EDuration	[1] unchangeable derived transient volatile	

endTime
 	
 tmsc.ITimeRange.
getEndTime()
	tmsc.ETimestamp	[1]
		unchangeable
		derived
		transient
		volatile	

projection	EBoolean		If true, this dependency was not originally part of the TMSC, but is an effect of projection, see nl.esi.pps.tmsc.util.TmscProjection class. Projections should typically not be visible to the user, unless they provide additional information.
scheduled	EBooleanObject		If true, this dependency was induced by the (software) platform as result of scheduling. If false, this dependency reflects a control or data dependency as programmed in code. By default Executions and LifelineSegments-without-activeExecution are considered to be scheduled dependencies, see TmscRefinements#refineWithCallStacks(Lifeline).
startTime tmsc.ITimeRange. getStartTime()	tmsc.ETimestamp	[1] unchangeable derived transient volatile	

Name	Type	Properties	Description
timeBound	tmsc.EDuration		The time-bound specifies the lowerbound for the duration of this dependency. This analysis attribute is used in e.g. critical-path and slack analysis.

References

Name	Type	Properties	Description
scopes	tmsc.ScopedTMSC *EOpposite:* dependencies	non-resolveProxies unordered transient	All dependencies of a child-scope ScopedTMSC should be contained by its parent-scope TMSC. Note that this is a derived 'many' relation. Though the return type 'EList' provides methods to alter the content, no altering method should be used and will throw an UnsupportedOperationException upon usage.
source	tmsc.Event *EOpposite:* fullScopeOutgoingDependencies	non-resolveProxies [1] transient	
target	tmsc.Event *EOpposite:* fullScopeIncomingDependencies	non-resolveProxies [1] transient	
tmsc	tmsc.FullScopeTMSC *EOpposite:* dependencies	**container** [1]	

Operations

Name	Aspect and Type	Properties	Description
abstract isEpochTime() □ tmsc.ITimeRange.isEpochTime()	*returns* EBoolean		

Used at

- tmsc.EntryEvent.fullScopeIncomingDependencies

- `tmsc.EntryEvent.fullScopeOutgoingDependencies`
- `tmsc.Event.fullScopeIncomingDependencies`
- `tmsc.Event.fullScopeOutgoingDependencies`
- `tmsc.ExitEvent.fullScopeIncomingDependencies`
- `tmsc.ExitEvent.fullScopeOutgoingDependencies`
- `tmsc.FullScopeTMSC.dependencies`
- `tmsc.ITMSC.getIncomingDependencies(event)`
- `tmsc.ITMSC.getOutgoingDependencies(event)`
- `tmsc.ITMSC.getDependencies()`
- `tmsc.ITMSC.getIncomingDependencies(event)`
- `tmsc.ITMSC.getOutgoingDependencies(event)`
- `tmsc.ScopedTMSC.dependencies`
- `tmsc.ITMSC.getIncomingDependencies(event)`
- `tmsc.ITMSC.getOutgoingDependencies(event)`
- `tmsc.ITMSC.getDependencies()`
- `tmsc.ITMSC.getIncomingDependencies(event)`
- `tmsc.ITMSC.getOutgoingDependencies(event)`

4.8.2.2. Class DomainDependency

Super-types

- `properties.PropertiesContainer`
- `tmsc.Dependency`
- `tmsc.ITimeRange`

4.8.2.3. Class EntryEvent

Super-types

- `properties.PropertiesContainer`
- `tmsc.Event`

References

Name	Type	Properties	Description
`execution` ▯ `tmsc.Event.getExecution()` ▯ `tmsc.Event.setExecution(Execution)`	`tmsc.Execution` *EOpposite:* `entry`	non-resolveProxies transient	

Used at

- `tmsc.Execution.entry`

4.8.2.4. Abstract Class Event

Super-types

- `properties.PropertiesContainer`

Sub-types

- `tmsc.EntryEvent`
- `tmsc.ExitEvent`

Attributes

Name	Type	Properties	Description
`timestamp`	`tmsc.ETimestamp`	`[1]`	
`traced`	`EBoolean`	`[1]` *Default:* `true`	

Containments

Name	Type	Properties	Description
`arguments`	`tmsc.FunctionArgumentMapEntry`	resolveProxies	

References

Name	Type	Properties	Description
`component`	`specified.Component`	`[1]`	
`fullScopeIncomingDependencies`	`tmsc.Dependency` *EOpposite:* `target`	non-resolveProxies	
`fullScopeOutgoingDependencies`	`tmsc.Dependency` *EOpposite:* `source`	non-resolveProxies	
`function`	`implemented.Function`	`[1]`	
`lifeline`	`tmsc.Lifeline` *EOpposite:* `events`	**container** `[1]`	

Name	Type	Properties	Description
`scopes`	`tmsc.ScopedTMSC`	non-resolveProxies unordered unchangeable derived transient volatile	Note that this is a derived 'many' relation. Though the return type 'EList' provides methods to alter the content, no altering method should be used and will throw an UnsupportedOperationException upon usage.
`tmsc`	`tmsc.FullScopeTMSC`	non-resolveProxies [1] unchangeable derived transient volatile	

Operations

Name	Aspect and Type	Properties	Description
abstract `getExecution()` ▯ `tmsc.EntryEvent.execution` ▯ `tmsc.ExitEvent.execution`	*returns* `tmsc.Execution`		
abstract `setExecution(Execution)` ▯ `tmsc.EntryEvent.execution` ▯ `tmsc.ExitEvent.execution`	*returns* `void`		
	`Execution` `tmsc.Execution`		

Used at

- `metric.MetricInstance.from`
- `metric.MetricInstance.to`
- `tmsc.Dependency.source`
- `tmsc.Dependency.target`
- `tmsc.DomainDependency.source`
- `tmsc.DomainDependency.target`
- `tmsc.ITMSC.getEvents()`
- `tmsc.ITMSC.getIncomingDependencies(event)`

- `tmsc.ITMSC.getOutgoingDependencies(event)`
- `tmsc.ITMSC.getEvents()`
- `tmsc.ITMSC.getIncomingDependencies(event)`
- `tmsc.ITMSC.getOutgoingDependencies(event)`
- `tmsc.Interval.from`
- `tmsc.Interval.to`
- `tmsc.Lifeline.events`
- `tmsc.LifelineSegment.source`
- `tmsc.LifelineSegment.target`
- `tmsc.Measurement.from`
- `tmsc.Measurement.to`
- `tmsc.Message.source`
- `tmsc.Message.target`
- `tmsc.MessageControl.source`
- `tmsc.MessageControl.target`
- `tmsc.Reply.source`
- `tmsc.Reply.target`
- `tmsc.Request.source`
- `tmsc.Request.target`
- `tmsc.ScopedTMSC.origin`
- `tmsc.ITMSC.getEvents()`
- `tmsc.ITMSC.getIncomingDependencies(event)`
- `tmsc.ITMSC.getOutgoingDependencies(event)`
- `tmsc.ITMSC.getEvents()`
- `tmsc.ITMSC.getIncomingDependencies(event)`
- `tmsc.ITMSC.getOutgoingDependencies(event)`

4.8.2.5. Class Execution

Execution should be considered final and should not be overridden. Executions are added by means of a refinement transformation, see {@link nl.esi.pps.tmsc.transform.TmscRefinements#refineWithCallStacks(FullScopeTMSC)}

Super-types

- `properties.PropertiesContainer`
- `tmsc.ITimeRange`

Attributes

Name	Type	Properties	Description
duration ▯ tmsc.ITimeRange. getDuration()	tmsc.EDuration	[1] unchangeable derived transient volatile	

endTime
 	
▯ tmsc.ITimeRange.
getEndTime()
	tmsc.ETimestamp	[1]
		unchangeable
		derived
		transient
		volatile	

startTime ▯ tmsc.ITimeRange. getStartTime()	tmsc.ETimestamp	[1] unchangeable derived transient volatile	

References

Name	Type	Properties	Description
children	tmsc.Execution *EOpposite:* parent	non- resolveProxies	
component	specified. Component	non- resolveProxies [1] unchangeable derived transient volatile	
entry	tmsc.EntryEvent *EOpposite:* execution	non- resolveProxies [1]	
exit	tmsc.ExitEvent *EOpposite:* execution	non- resolveProxies [1]	

Name	Type	Properties	Description
function	implemented.Function	non-resolveProxies [1] unchangeable derived transient volatile	
lifeline	tmsc.Lifeline *EOpposite:* executions	non-resolveProxies **container** [1] transient	
parent	tmsc.Execution *EOpposite:* children	non-resolveProxies transient	Definition 2.3.5
root	tmsc.Execution	non-resolveProxies [1] unchangeable derived transient volatile	
segments	tmsc.LifelineSegment *EOpposite:* activeExecution	non-resolveProxies unchangeable derived transient volatile	Returns the LifeLineSegments for which this Execution is the active execution on its LifeLine. Note that this is a derived 'many' relation. Though the return type 'EList' provides methods to alter the content, no altering method should be used and will throw an UnsupportedOperationException upon usage.
tmsc	tmsc.FullScopeTMSC	non-resolveProxies [1] unchangeable derived transient volatile	

Operations

Name	Aspect and Type	Properties	Description
abstract isEpochTime() tmsc.ITimeRange. isEpochTime()	*returns* EBoolean		

Used at

- tmsc.EntryEvent.execution
- tmsc.Event.getExecution()
- tmsc.Event.setExecution(Execution)
- tmsc.Execution.children
- tmsc.Execution.parent
- tmsc.Execution.root
- tmsc.ExitEvent.execution
- tmsc.Lifeline.executions
- tmsc.Lifeline.rootExecutions
- tmsc.LifelineSegment.activeExecution

4.8.2.6. Class ExitEvent

Super-types

- properties.PropertiesContainer
- tmsc.Event

References

Name	Type	Properties	Description
execution tmsc.Event. getExecution() tmsc.Event. setExecution(Execution)	tmsc.Execution *EOpposite:* exit	non-resolveProxies transient	

Used at

- tmsc.Execution.exit

4.8.2.7. Class FullScopeTMSC

FullScopeTMSC contains all events (via Lifeline) and dependencies of the TMSC model. It has the startTime and endTime attributes which are timestamps derived from the TMSC model, denoting the start and end time of the considered trace.

Super-types

- properties.PropertiesContainer
- tmsc.ITMSC
- tmsc.ITimeRange
- tmsc.TMSC

Attributes

Name	Type	Properties	Description
duration ▯ tmsc.ITimeRange.getDuration()	tmsc.EDuration	[1] unchangeable derived transient volatile	
endTime ▯ tmsc.ITimeRange.getEndTime()	tmsc.ETimestamp	[1]	
epochTime	EBoolean	[1] *Default:* false	
startTime ▯ tmsc.ITimeRange.getStartTime()	tmsc.ETimestamp	[1]	

Containments

Name	Type	Properties	Description
dependencies ▯ tmsc.ITMSC.getDependencies()	tmsc.Dependency *EOpposite:* tmsc		
lifelines	tmsc.Lifeline *EOpposite:* tmsc		
measurements	tmsc.Measurement	resolveProxies	

References

Name	Type	Properties	Description
architectures	architecture.ArchitectureModel		References all model roots (i.e. ArchitectureModels) that own the architectural elements that are referenced by the model elements of this FullScopeTMSC.

Used at

- `metric.Metric.tmscs`
- `metric.MetricInstance.tmsc`
- `tmsc.Dependency.tmsc`
- `tmsc.DomainDependency.tmsc`
- `tmsc.EntryEvent.tmsc`
- `tmsc.Event.tmsc`
- `tmsc.Execution.tmsc`
- `tmsc.ExitEvent.tmsc`
- `tmsc.FullScopeTMSC.fullScope`
- `tmsc.Interval.tmsc`
- `tmsc.Lifeline.tmsc`
- `tmsc.LifelineSegment.tmsc`
- `tmsc.Measurement.tmsc`
- `tmsc.Message.tmsc`
- `tmsc.MessageControl.tmsc`
- `tmsc.Reply.tmsc`
- `tmsc.Request.tmsc`
- `tmsc.ScopedTMSC.fullScope`
- `tmsc.TMSC.fullScope`

4.8.2.8. Class FunctionArgumentMapEntry

Instance Type Name

`java.util.Map$Entry`

Attributes

Name	Type	Properties	Description
`value`	`EString`	`[1]`	For now we have chosen to persist function arguments as a String to avoid the introduction of a complex type system.

References

Name	Type	Properties	Description
`key`	`implemented.FunctionParameter`	`[1]`	

Used at

- `tmsc.EntryEvent.arguments`

- `tmsc.Event.arguments`
- `tmsc.ExitEvent.arguments`

4.8.2.9. Interface ITMSC

Instance Type Name

`nl.esi.pps.tmsc.ITMSC`

Sub-types

- `tmsc.FullScopeTMSC`
- `tmsc.ScopedTMSC`
- `tmsc.TMSC`

Operations

Name	Aspect and Type	Properties	Description
abstract `getDependencies()` ▯ `tmsc.FullScopeTMSC.dependencies` ▯ `tmsc.ScopedTMSC.dependencies`	*returns* `tmsc.Dependency`	unique	
abstract `getEvents()`	*returns* `tmsc.Event`	unique	
abstract `getIncomingDependencies(event)`	*returns* `tmsc.Dependency`	unique	
	`event` `tmsc.Event`	`[1]`	
abstract `getOutgoingDependencies(event)`	*returns* `tmsc.Dependency`	unique	
	`event` `tmsc.Event`	`[1]`	

4.8.2.10. Interface ITimeRange

Instance Type Name

`nl.esi.pps.tmsc.ITimeRange`

Sub-types

- `metric.MetricInstance`
- `tmsc.Dependency`
- `tmsc.DomainDependency`

- tmsc.Execution
- tmsc.FullScopeTMSC
- tmsc.Interval
- tmsc.LifelineSegment
- tmsc.Measurement
- tmsc.Message
- tmsc.MessageControl
- tmsc.Reply
- tmsc.Request

Operations

Name	Aspect and Type	Properties	Description
abstract getDuration() ▯ tmsc.Dependency.duration ▯ tmsc.Execution.duration ▯ tmsc.FullScopeTMSC.duration ▯ tmsc.Interval.duration	*returns* tmsc.EDuration	[1]	
abstract getEndTime() ▯ tmsc.Dependency.endTime ▯ tmsc.Execution.endTime ▯ tmsc.FullScopeTMSC.endTime ▯ tmsc.Interval.endTime	*returns* tmsc.ETimestamp	[1]	

Name	Aspect and Type	Properties	Description
abstract getStartTime() ▯ tmsc.Dependency.startTime ▯ tmsc.Execution.startTime ▯ tmsc.FullScopeTMSC.startTime ▯ tmsc.Interval.startTime	*returns* tmsc.ETimestamp	[1]	
abstract isEpochTime() ▯ tmsc.Dependency.isEpochTime() ▯ tmsc.Execution.isEpochTime() ▯ tmsc.Interval.isEpochTime()	*returns* EBoolean	[1]	

4.8.2.11. Abstract Class Interval

Super-types

- properties.PropertiesContainer
- tmsc.ITimeRange

Sub-types

- metric.MetricInstance
- tmsc.Measurement

Attributes

Name	Type	Properties	Description
duration ▯ tmsc.ITimeRange.getDuration()	tmsc.EDuration	[1] unchangeable derived transient volatile	

Name	Type	Properties	Description

endTime
 	
 tmsc.ITimeRange.
getEndTime()
	tmsc.ETimestamp	[1]
		unchangeable
		derived
		transient
		volatile	

startTime tmsc.ITimeRange. getStartTime()	tmsc.ETimestamp	[1] unchangeable derived transient volatile	

References

Name	Type	Properties	Description
from	tmsc.Event	[1]	
scopes	tmsc.ScopedTMSC		Typically analysis stores its result as a ScopedTMSC. This relation can be used to avoid duplicate analysis results in the model.
tmsc	tmsc. FullScopeTMSC	[1] unchangeable derived transient volatile	
to	tmsc.Event	[1]	

Operations

Name	Aspect and Type	Properties	Description
abstract getName() metric.MetricInstance. name tmsc.Measurement.name	*returns* EString	[1]	
abstract isEpochTime() tmsc.ITimeRange. isEpochTime()	*returns* EBoolean		

4.8.2.12. Class Lifeline

Lifeline is the modeling element as defined in the UML Message Sequence Chart formalism and it is equivalent only to the instantiated nature (viewpoint) of the Component modeling element as defined in the abstract TMSC meta-model. A lifeline contains the behavior of an Executor, in the form of a sequence of events, over a specific time span.

Super-types

- properties.PropertiesContainer

Attributes

Name	Type	Properties	Description
traced	EBoolean	[1] unchangeable derived transient volatile	

Containments

Name	Type	Properties	Description
events	tmsc.Event *EOpposite:* lifeline		
executions	tmsc.Execution *EOpposite:* lifeline		

References

Name	Type	Properties	Description
executor	instantiated. Executor	[1]	
rootExecutions	tmsc.Execution	non-resolveProxies unchangeable derived transient volatile	
segments	tmsc. LifelineSegment *EOpposite:* lifeline	non-resolveProxies unchangeable derived transient volatile	Note that this is a derived 'many' relation. Though the return type 'EList' provides methods to alter the content, no altering method should be used and will throw an UnsupportedOperationException upon usage.

Name	Type	Properties	Description
tmsc	tmsc.FullScopeTMSC *EOpposite:* lifelines	**container** [1]	

Used at

- tmsc.EntryEvent.lifeline
- tmsc.Event.lifeline
- tmsc.Execution.lifeline
- tmsc.ExitEvent.lifeline
- tmsc.FullScopeTMSC.lifelines
- tmsc.LifelineSegment.lifeline

4.8.2.13. Class LifelineSegment

Super-types

- properties.PropertiesContainer
- tmsc.Dependency
- tmsc.ITimeRange

References

Name	Type	Properties	Description
activeExecution	tmsc.Execution *EOpposite:* segments	non-resolveProxies unchangeable derived transient volatile	
lifeline	tmsc.Lifeline *EOpposite:* segments	non-resolveProxies [1] unchangeable derived transient volatile	

Used at

- tmsc.Execution.segments
- tmsc.Lifeline.segments

4.8.2.14. Class Measurement

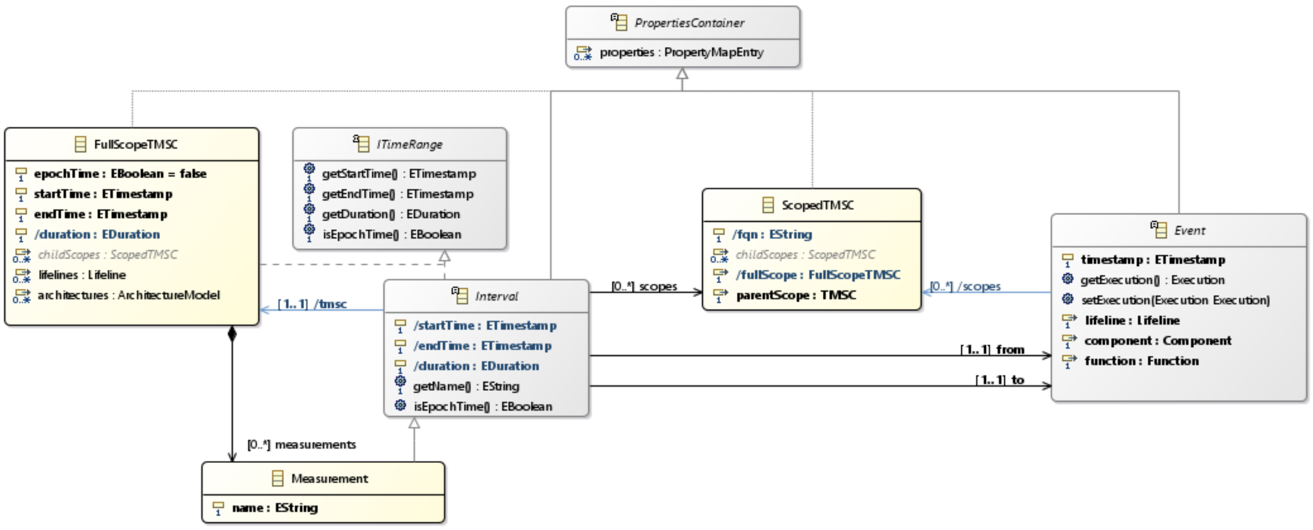


Figure 11. Diagram of measurements in the TMSC meta-model

Super-types

- `properties.PropertiesContainer`
- `tmsc.ITimeRange`
- `tmsc.Interval`

Attributes

Name	Type	Properties	Description

`name`
 	
`tmsc.Interval.getName()`
	`EString`	`[1]`	

Used at

- `tmsc.FullScopeTMSC.measurements`

4.8.2.15. Class Message

Super-types

- `properties.PropertiesContainer`
- `tmsc.Dependency`
- `tmsc.ITimeRange`

Sub-types

- `tmsc.Reply`
- `tmsc.Request`

References

Name	Type	Properties	Description
controlDependencies	tmsc.MessageControl *EOpposite:* message	non-resolveProxies	

Used at

- tmsc.MessageControl.message

4.8.2.16. Class MessageControl

Super-types

- properties.PropertiesContainer
- tmsc.Dependency
- tmsc.ITimeRange

References

Name	Type	Properties	Description
message	tmsc.Message *EOpposite:* controlDependencies	non-resolveProxies [1] transient	

Used at

- tmsc.Message.controlDependencies
- tmsc.Reply.controlDependencies
- tmsc.Request.controlDependencies

4.8.2.17. Class Reply

Super-types

- properties.PropertiesContainer
- tmsc.Dependency
- tmsc.ITimeRange
- tmsc.Message

References

Name	Type	Properties	Description
request	tmsc.Request *EOpposite:* replies	non-resolveProxies transient	

Used at

- `tmsc.Request.replies`

4.8.2.18. Class Request

Super-types

- `properties.PropertiesContainer`
- `tmsc.Dependency`
- `tmsc.ITimeRange`
- `tmsc.Message`

References

Name	Type	Properties	Description
`replies`	`tmsc.Reply` *EOpposite:* `request`	non-resolveProxies	

Used at

- `tmsc.Reply.request`

4.8.2.19. Class ScopedTMSC

ScopedTMSC is a sub-graph of the whole TMSC model and defined for facilitating fragmentation of the TMSC model to sub-graphs (e.g. indication of a path), i.e. events and dependencies that hold specific properties, activities, etc. Each ScopedTMSC can be considered as a part of its parent scope TMSC and allows application of analysis techniques, i.e., critical path analysis, locally in a sub-graph. The dependencies of a child scope are contained by its parent scope.

Super-types

- `properties.PropertiesContainer`
- `tmsc.ITMSC`
- `tmsc.TMSC`

Attributes

Name	Type	Properties	Description
`fqn`	`EString`	`[1]` unchangeable derived transient volatile	
`name`	`tmsc.EID`	`[1]`	

References

Name	Type	Properties	Description
dependencies tmsc.ITMSC. getDependencies()	tmsc.Dependency *EOpposite:* scopes	non-resolveProxies	
origin	tmsc.Event		Typically a ScopedTMSC represents an analysis result that is created from a certain event or events, this is called the origin of the scope. Setting this attribute is optional, but may help successor analyses.
parentScope	tmsc.TMSC *EOpposite:* childScopes	**container** [1]	

Used at

- metric.MetricInstance.scopes
- tmsc.Dependency.scopes
- tmsc.DomainDependency.scopes
- tmsc.EntryEvent.scopes
- tmsc.Event.scopes
- tmsc.ExitEvent.scopes
- tmsc.FullScopeTMSC.childScopes
- tmsc.Interval.scopes
- tmsc.LifelineSegment.scopes
- tmsc.Measurement.scopes
- tmsc.Message.scopes
- tmsc.MessageControl.scopes
- tmsc.Reply.scopes
- tmsc.Request.scopes
- tmsc.ScopedTMSC.childScopes
- tmsc.TMSC.childScopes

4.8.2.20. Abstract Class TMSC

TMSC is a directed acyclic graph with events and dependencies, and some additional structural properties. A TMSC should at least contain all events that are referred to by its dependencies. Two specializations are considered the FullScopeTMSC and ScopedTMSC.

Super-types

- `properties.PropertiesContainer`
- `tmsc.ITMSC`

Sub-types

- `tmsc.FullScopeTMSC`
- `tmsc.ScopedTMSC`

Containments

Name	**Type**	**Properties**	**Description**
`childScopes`	`tmsc.ScopedTMSC` *EOpposite:* `parentScope`	*EKeys:* `name`	All dependencies of a child scope should also be contained by its parent scope.

References

Name	**Type**	**Properties**	**Description**
`fullScope`	`tmsc.FullScopeTMSC`	non-resolveProxies [1] unchangeable derived transient volatile	

Used at

- `tmsc.ScopedTMSC.parentScope`

[1] H. Koziolek, "Performance Evaluation of Component-based Software Systems: A Survey", Elsevier, 2010.