2017

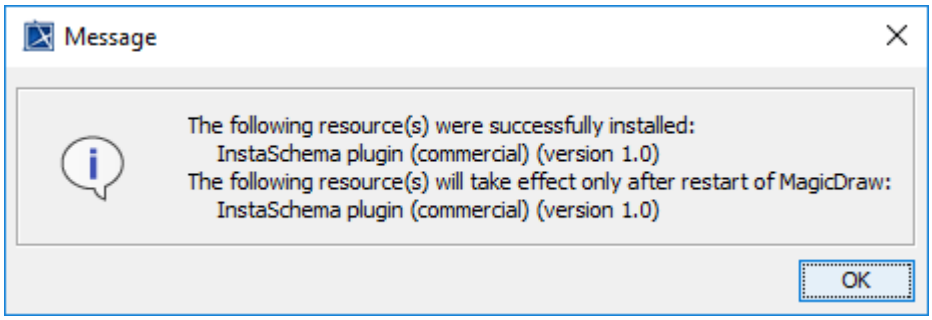
InstaSchema Quick-Start Guide (Version 1.2)

Table of Contents

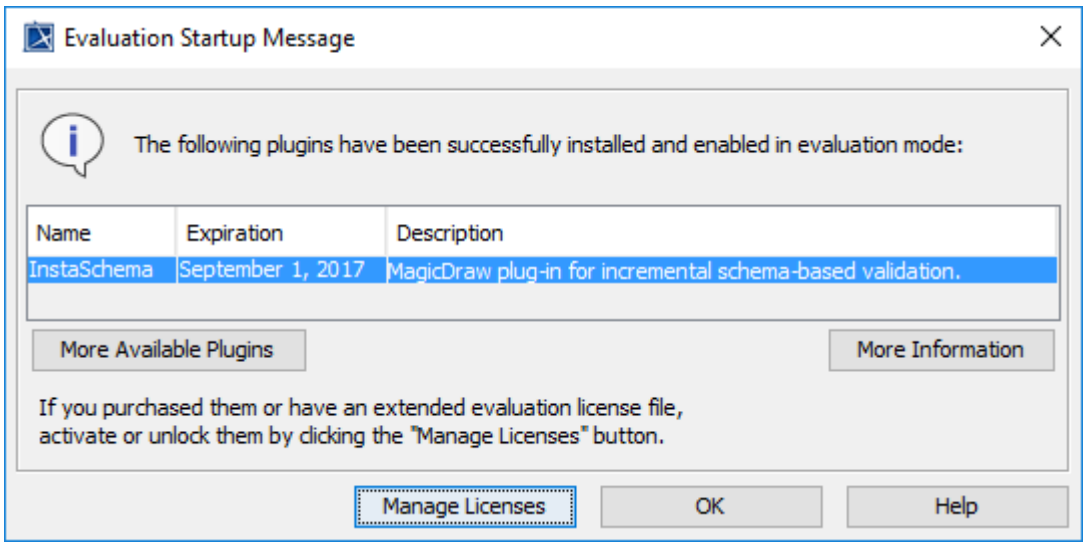
Installation Guide

Installation of the InstaSchema MagicDraw plug-in is straightforward and is done in several simple steps:

1. Install MagicDraw 18.5
2. Install MagicDraw SysML plug-in (Optional)
3. Download InstaSchema archive file
4. Open and anctivate MagicDraw
5. Open Resource/Plugin Manager (Help → Resource/Plugin Manager)
6. Select 'Import'
7. Navigate to the downloaded InstaSchema archive and select it
8. If the installation is successful, the following notification should appear:



9. Restart MagicDraw, read and accept the InstaSchema EULA. After this, the following evaluation startup message should appear.



10. At this point, the installation is complete. By default the InstaSchema evaluation version can be used for up to 90 days.

Basic Concepts

Profile Definitions in MagicDraw

In MagicDraw, it is possible to extend built-in UML and SysML profiles, through custom profile definition and DSL development. These profiles consist of sets of Stereotype objects, each associated with a meta-class. Stereotypes with a certain meta-class can be applied to instances of that given type, to extend them with domain specific information. this information is stored in tagged values. More information on profiling can be found in the MagicDraw UML Profiling and DSL Guide.

DSL Customizations

MagicDraw profiles define sets of stereotypes that extends the UML metamodel, while customizations focus on auxiliary and tooling related information for each stereotype of a given profile. Through these customizations, UML profiles can be expanded into fully fledged domain-specific languages. these customization objects define among others, the following:

- Containment constraints
- Endpoint type constraints for edges
- Also allows the definition of smart tagged value instantiation..

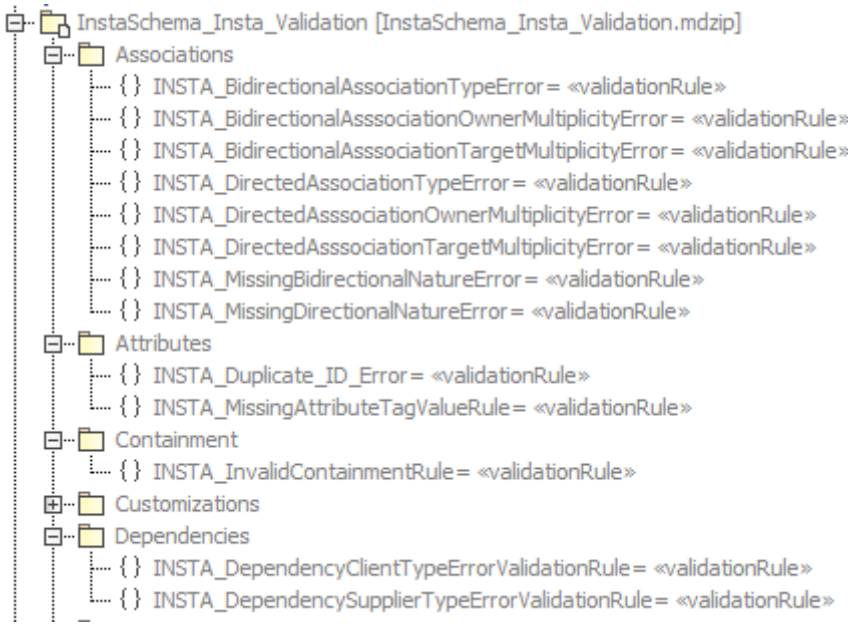
After the profile and its corresponding customizations have been defined, the profile's domain specific types can be used in custom diagram types as well. More information on domain specific modelling can be found in the MagicDraw UML Profiling and DSL Guide.

Structure-based validation in InstaSchema

As with any modeling language, custom DSLs need to be validated. DSL validation can be achieved via sets of user defined UML constraints. Large sets of these, however can be tedious to define. InstaSchema automatically defines conformance validation rules based on DSL structure.

InstaSchema introduces the *SchemaCustomization* stereotype, which can be applied to *Customization* objects to mark them for the validation engine. Elements with the given customization's target stereotype are validated based on the information stored in the profile definition and the appropriate customizations. *SchemaCustomization* stereotype instances also contain information regarding additional *Association*-related constraints.

The InstaSchema validation engine contains a set of predefined constraints, which are parameterized based on the contents of the profile definition. These rules are contained in a shared project shipped with InstaSchema.

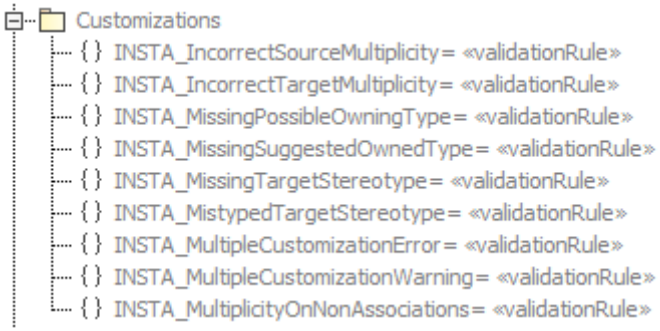


1. Figure Profile validation rules in InstaSchema

The validation rule set defines the following structural validation rules:

- Rules for edges in the DSL:
  - Stereotyped Association: Role types shall correspond to the types specified in the appropriate customization.
  - Stereotyped Dependency: Endpoint types shall correspond to the types specified in the appropriate customization.
  - Stereotyped Association: Role multiplicity shall conform with the specified tag values in the related *SchemaCustomization* instance.
  - Stereotyped Association: Role navigability shall conform with the specified tag values in the related *SchemaCustomization* instance
- Rules for DSL Properties (Tag Values)
  - There shall be no duplicate tag value of an ID Propery (isID == true).
  - There shall be no missing tag value for a mandatory stereotype Property (multiplicity > 0).
- Rules for containment relations in DSLs
  - Stereotyped element: Each stereotyped element shall be contained by an element specified in its customization's *possible Owners* property.

Naturally, during the definition of the customizations themselves possible errors might remain hidden. To abolish this issue, InstaSchema defines a set of *meta-level* rules that check whether the customization definitions contain errors.



2. Figure Meta validation rules in InstaSchema

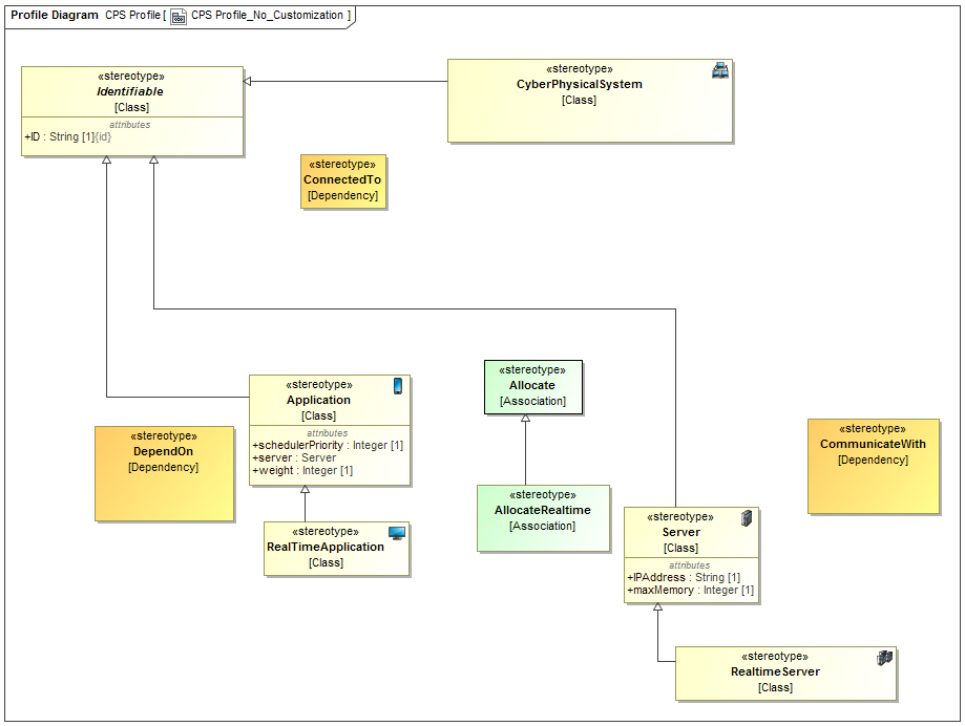
Both meta, and instance level rule implementations field quick fixes, to help the quick resolution of DSL-related validation errors. As the validation rules are only parameterized by the input profile and customization data, the rules themselves are already implemented. Based on the input paremeters (DSL elements), it is possible for the framework to provide sane quick fix suggestions for each pre-defined validation rule.

Cyber-Physical Systems Example

The following part elaborates how the InstaSchema validation is enabled on an already existing DSL. For this part, please download the InstaSchema example zip file, and open the project *SysMLSmartHouseDemo.mdzip*. Note, that the archive also contains a version of the project where InstaSchema validation has already been set up.

Example project - CPS Profile -

the example project *SysMLSmartHouseDemo* defines a custom DSL for cyber-physical systems. Cyber-physical systems are a network of servers that host different applications, which need to be able to communicate with each other. These servers and applications also possess domain specific properties. The figure below shows the UML profile used in the example project.



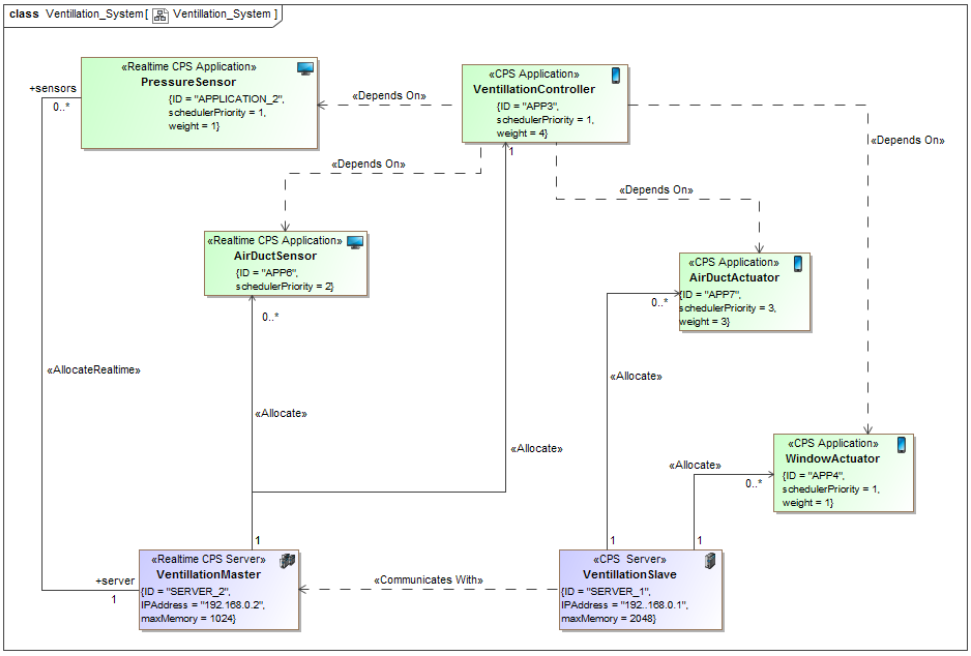
3. Figure: The CPS profile

The profile consists of the following stereotypes:

- ***Identifiable***: Abstract supertype of every CPS type, defines a mandatory String ID property.
  - metaClass: Class
- **CyberPhysicalSystem**: Main unit of containment, can contain variaous servers and applications.
  - metaClass: Class
- **ConnectedTo**: Connection relation defined between two CyberPhysicalSystem elements.
  - metaClass: Dependency
- **Application**: Base application type
  - metaClass: Class
  - Defined properties:
    - schedulerPriority : Integer [1]: Mandatory integer attribute that defines the priority of each application in the execution queue.
    - server : Server: Optional property that stores the server allocating the Application

    - weigth: Integer [1]: Mandatory integer property that defines the server resources required for running the given application.
- **RealTimeApplication**: Subtype of Application, defines an application that can only be run on real time servers.
- **DependOn**: Relation between two Application elements, meaning that one application relies on the output of another.
    - metaClass: Dependency
- **Server**: Base server type
    - metaClass: Class
    - Defined properties:
        - IPAddress: String [1]: Mandatory String attribute that contains the server's IP range
        - maxMemory: Integer [1]: Mandatory Integer attribute that defines the maximum available memory for the given server.
- **RealTimeServer**: Server that can also host RealTimeApplications
- **CommunicateWith**: Defines communication capabilities between two server objects.
    - metaClass: Dependency
- **Allocate**: Stereotyped association that specifies which server hosts which application
    - metaClass: Association
- **AllocateRealtime**: Subtype of Allocate, defines which RealtimeServer hosts which application.

The Example project also defines a DSL for this profile via a set of Customization objects and domain specific diagrams. the project also contains an instance CPS model describing parts of a simple Smart Home system.



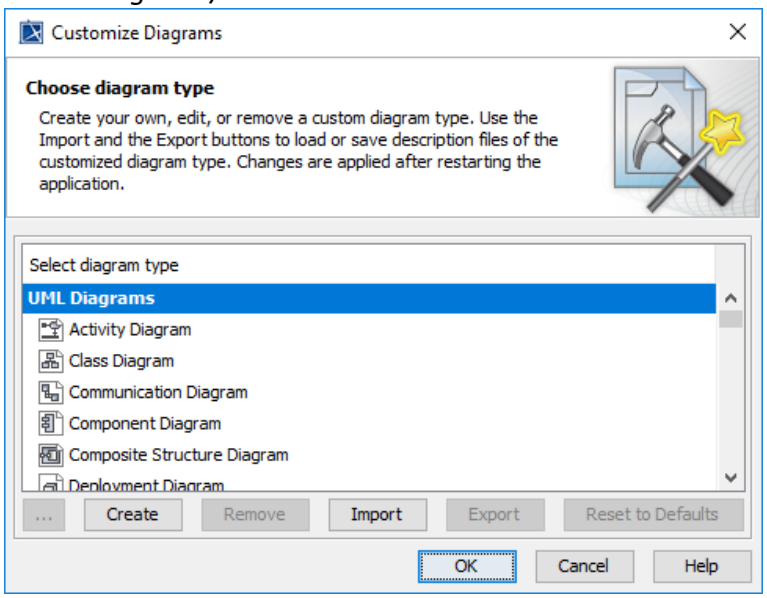
4. Figure: ventillation sub-system of the Smart House model

Enable Structure-based validation for the CPS DSL

To enable structure-based validation for the CPS DSL, the following steps need to be undertaken:

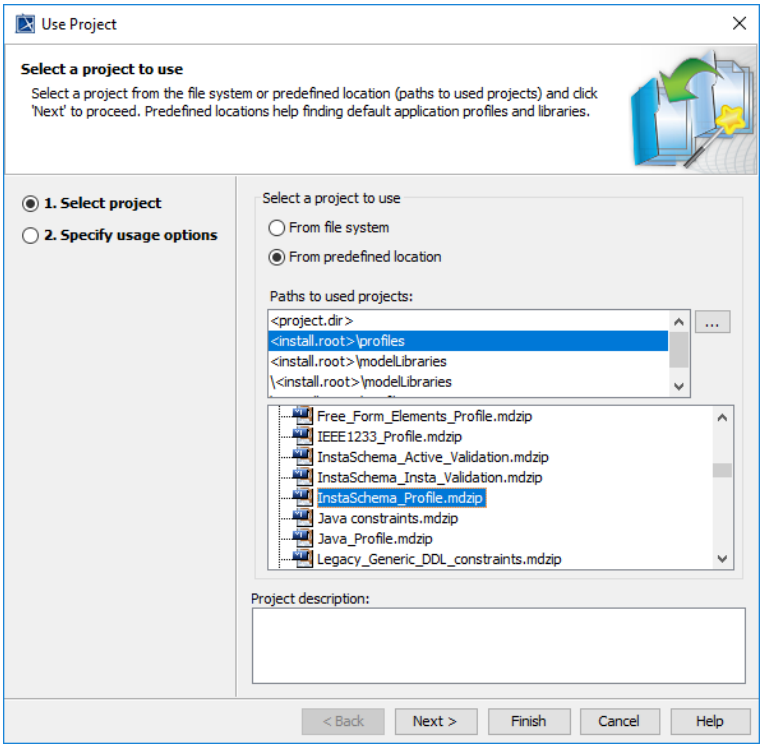
1. Install InstaSchema, and open the SysMLSmartHouseDemo project.

2. If not imported automatically, import the diagrams specified by the CPS DSL, via the menu *Diagrams/Customize...*



   a. Select *Import* and navigate to *CPS_Detail_Diagram_descriptor.xml* and *CPS_Overview_Diagram_descriptor.xml*

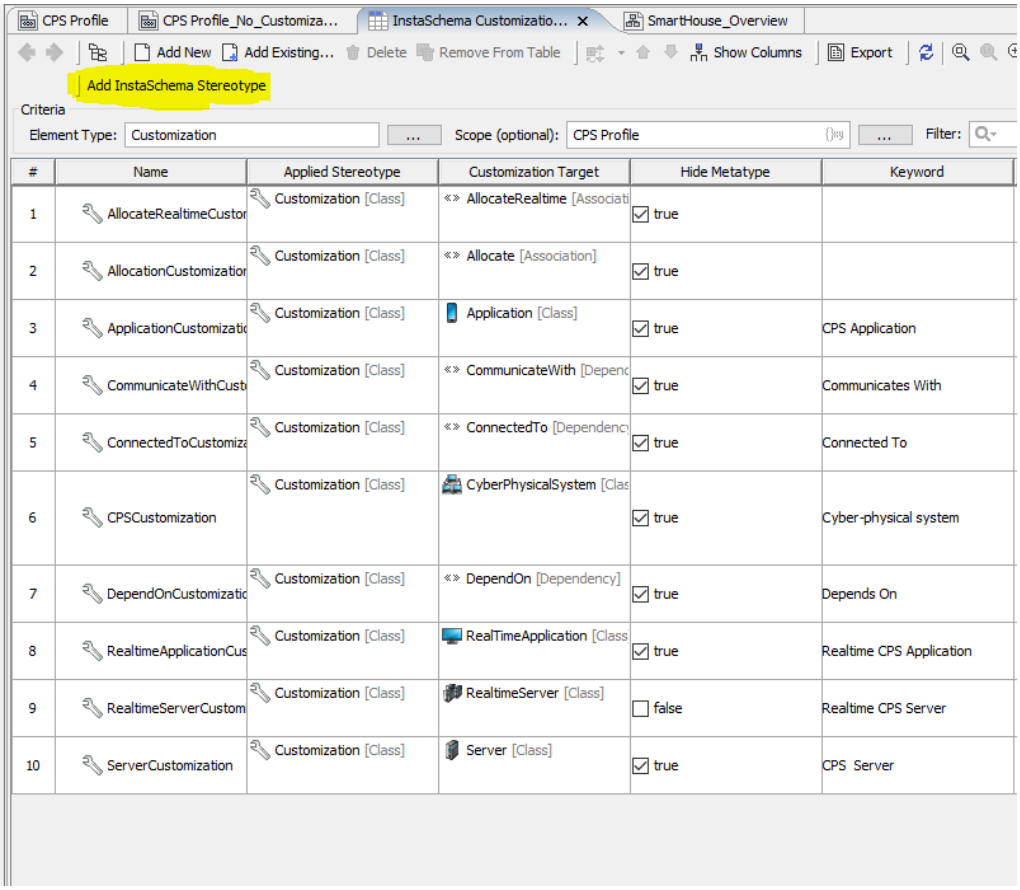
3. Select the InstaSchema Profile as external resource, via File/Use Project/Use Local Project
   a. Here Select InstaSchemaProfile as shown on the figure below



   b. Also select InstaSchema_Insta_Validation in the similar fashion.

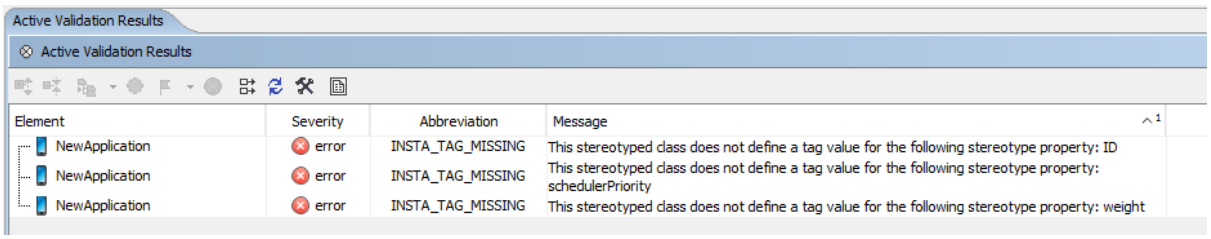
4. Navigate to the *CPS Profile* package, and select 'Define InstaSchema Customization Model' from the right click context menu
5. At this point a generic table opens that summarizes the already implemented customizations for the CPS DSL.

a. Add *SchemaCustomization* stereotype to the customizations, to enable InstaSchema validation on them. This can be done via the button *'Add InstaSchema Stereotype'*



b. From this point on the validation and quick-fix features of InstaSchema are available.

6. At this point the validation is enabled, Open the CPS detail diagram *Ventillation_System*
   a. Add a new Application element to it
   b. The following errors should appear:



Thank you for trying InstaSchema. Keep in mind however, that the tool itself is still experimental. If you encounter any issues please report them on the following forum topic.

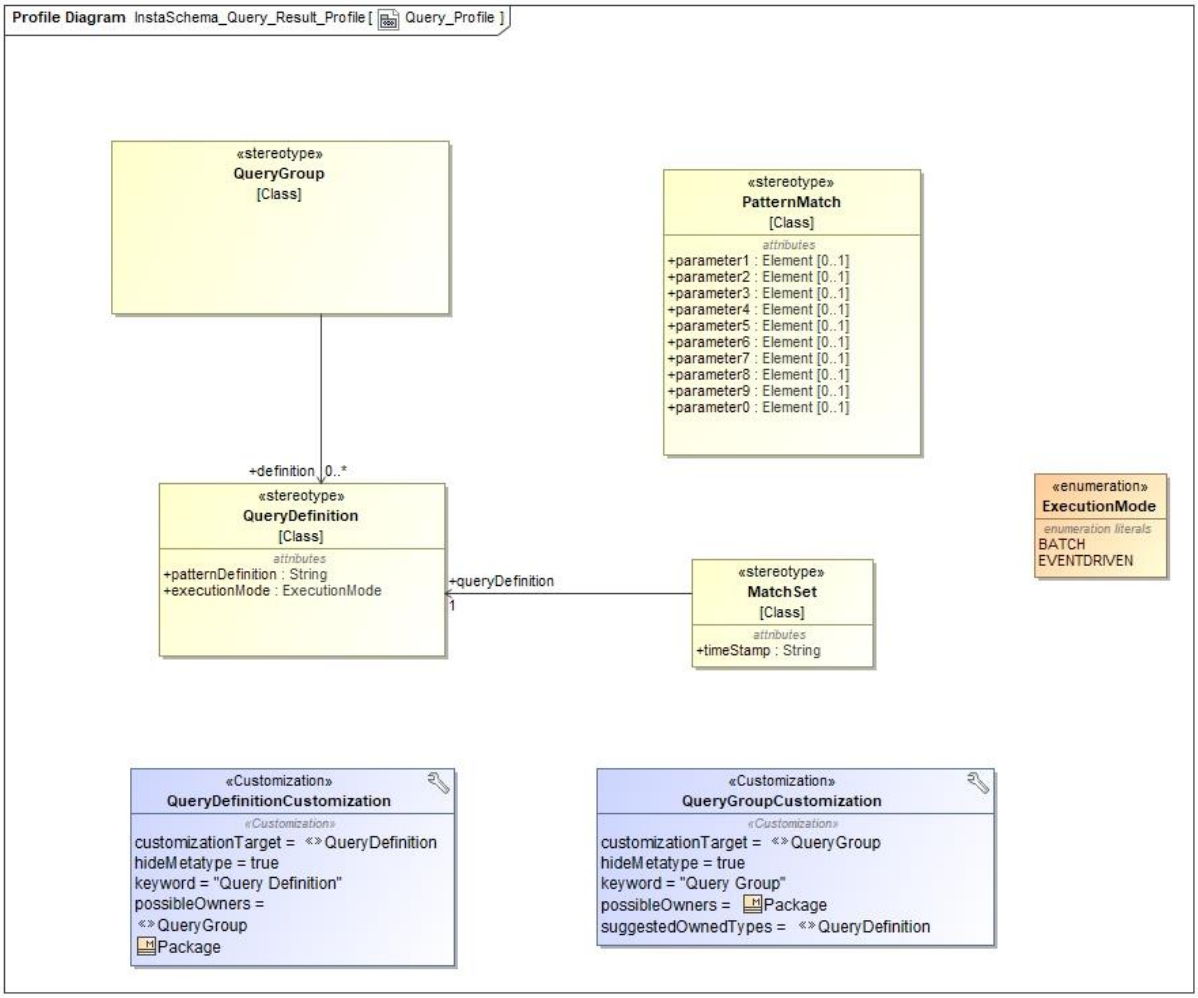
Custom Query Evaluation

In InstaSchema 1.1 the evaluation of custom queries is being introduced. To fit in with the UML-based modeling environment, both query definitions and query results are represented as

stereotyped model elements. This way the user can define the queries and aggregate results via using simple model operations.
InstaSchema 1.1 also introduces a generic table-based view for visualizing query results in a streamlined fashion.

Query Definition Profile

As mentioned before queries and their results are represented as stereotyped UML elements. To this end, the *InstaSchema Query Definition Profile* is introduced. It allows the user to define and group model queries, and instances of this profile are also used by InstaSchema to store the results of the queries themselves.



Profile Element Summary:

- QueryGroup: Defines groups of queries that can be executed in a batch operation.
- QueryDefinition: Contains the textual definition of the query, and the desired execution mode as well.
    - Textual definition: String containing the query written in the VIATRA Query language syntax.
    - Execution mode: defines how the query should be executed:
        - Batch: Typical run-once type execution, for each execution, a new result set is created

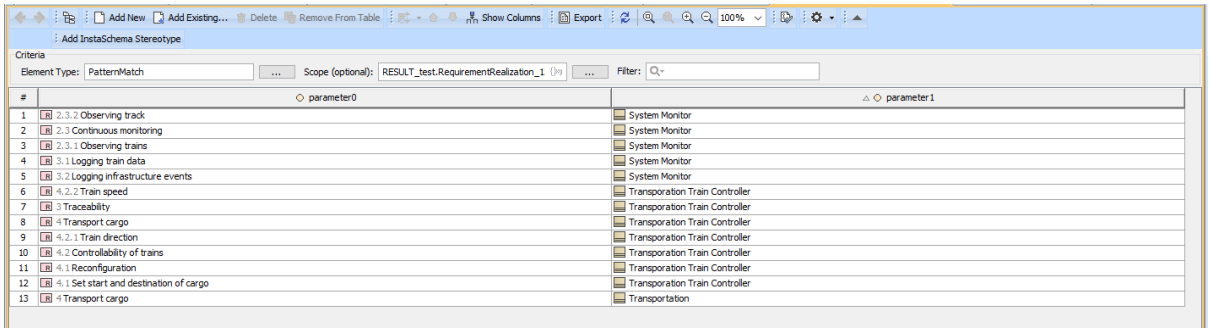
    - Event-driven: Result sets of queries are maintained automatically as the model changes
    - NOTE: In the current version only the Batch execution mode is available.
- MatchSet: Represents the results af a given query evaluation. Contains a set of PatternMatch objects.
- PatternMatch: Represents an individual match of a VIATRA query. It references elements that match the given query, as parameters.
    - NOTE: In order to provide a generic table based representation of pattern matches, the number of parameters need to be limited. In the current version this limit is 10.
- NOTE: As query results are represented as UML model elements, currently query parameters that have plain Java types (String instead of UML String Literal) are disallowed.

The VIATRA Query Language

InstaSchema 1.1 uses the VIATRA Query Language (VQL) syntax to define model queries. The basics of pattern definition and pattern language syntax I available on the VIATRA wiki: https://wiki.eclipse.org/VIATRA/Query/UserDocumentation/QueryLanguage

Query Result Diagram

InstaSchema introduces a generic table responsible for visualizing query results. An example table can be observed on the figure below.



Each row of the table represents a set of model objects that match the pattern. The columns of the table represent the parameters of the query.

Using the Custom Query Evaluation

- Create a new Sysml project.
- Use the InstaSchema Query Result Profile.
- Define the SysML model.
- Create a package for queries (not necessary but helps with organization).
- Add a query group (again not necessary but helps with grouping the queries).
- Add a query definition to the group.
    - Set the execution mode attribute to BATCH.
    - Define the query using the VQL syntax.
    - Keep in mind the following constraints:
        - Do not omit the package definition, as it is a required element of a VQL query definition. However due to the fact that the query is being

executed on-the-fly in MagicDraw, the actual package name will not be used.
    - Query patterns should have no more than 10 parameters in the current version.
    - Pattern parameters should use the UML wrappers for primitives (E.g: StringLiteral for String).
    - In the current version only the BATCH execution mode is supported.
- Right-click the query group (or definition).
    - Select 'Execute Selected VIATRA Queries'.
- A new match set is created
    - A generic table is opened, visualizing the query results.

Thank you for trying InstaSchema. Keep in mind however, that the tool itself is still experimental. If you encounter any issues please report them on the following forum topic.

Custom Validation Rule Definition

InstaSchema 1.2 allows the user to define custom validation rules through VIATRA query definitions. These query definitions can either be added as model elements, or defined as classes of a plug-in that depends on the newly introduced InstaSchema validation rule API.

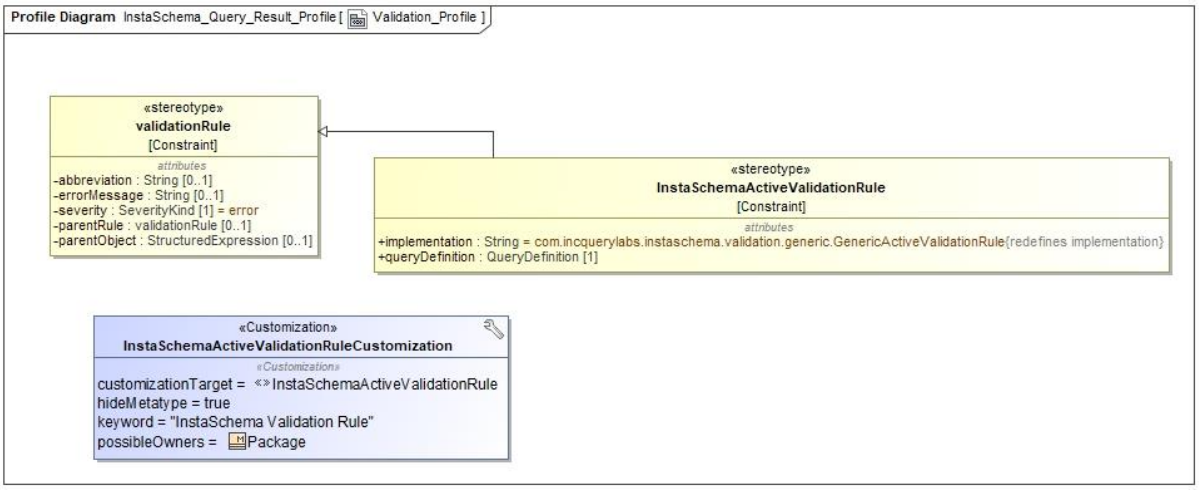
Defining Rules as Model Elements

Since InstaSchema 1.1 VIATRA query definitions can be added to MagicDraw models and executed in the tool's domain as well. These elements allow the definition of model queries using the VIATRA query language. Instaschema 1.2 allows the definition of validation rules in MD models, highly relying on the custom query definitions. These rules can be defined via instantiating elements of the newly introduced validation rule extension of the custom query definition profile.

The rules themselves are MagicDraw active validation rules that rely on a generic rule implementation. This generic implementation executes the VIATRA patterns specified in a query definition and creates annotations for its matches. The severity and message of these annotations is specified in the validation rule object itself.

Validation Rule Profile

In InstaSchema 1.2, the custom query definition profile has been expanded with the following elements:



InstaSchemaActiveValidationRule: This stereotype is a child stereotype of the *validationRule* stereotype, which means that constraints with the *InstaSchemaActiveValidationRule* stereotype can be handled by the MagicDraw active validation framework.

The stereotype defines the following new properties:

- Implementation: A property that redefines the similarly-named property of the *rule* stereotype. This way each created instaSchema active validation rule will refer the same generic rule implementation.
- QueryDefinition: A property that references the *QueryDefinition* object that specifies the query responsible for implementing the validation rule logic.

Properties inherited from *ValidationRule*:

- Message: The string value specified in the *message* property of *validationRule* stereotype instances, will be used as the annotation messages.
- Severity: The severity value of the given InstaSchema active validation rule will set the severity of the produced annotations.

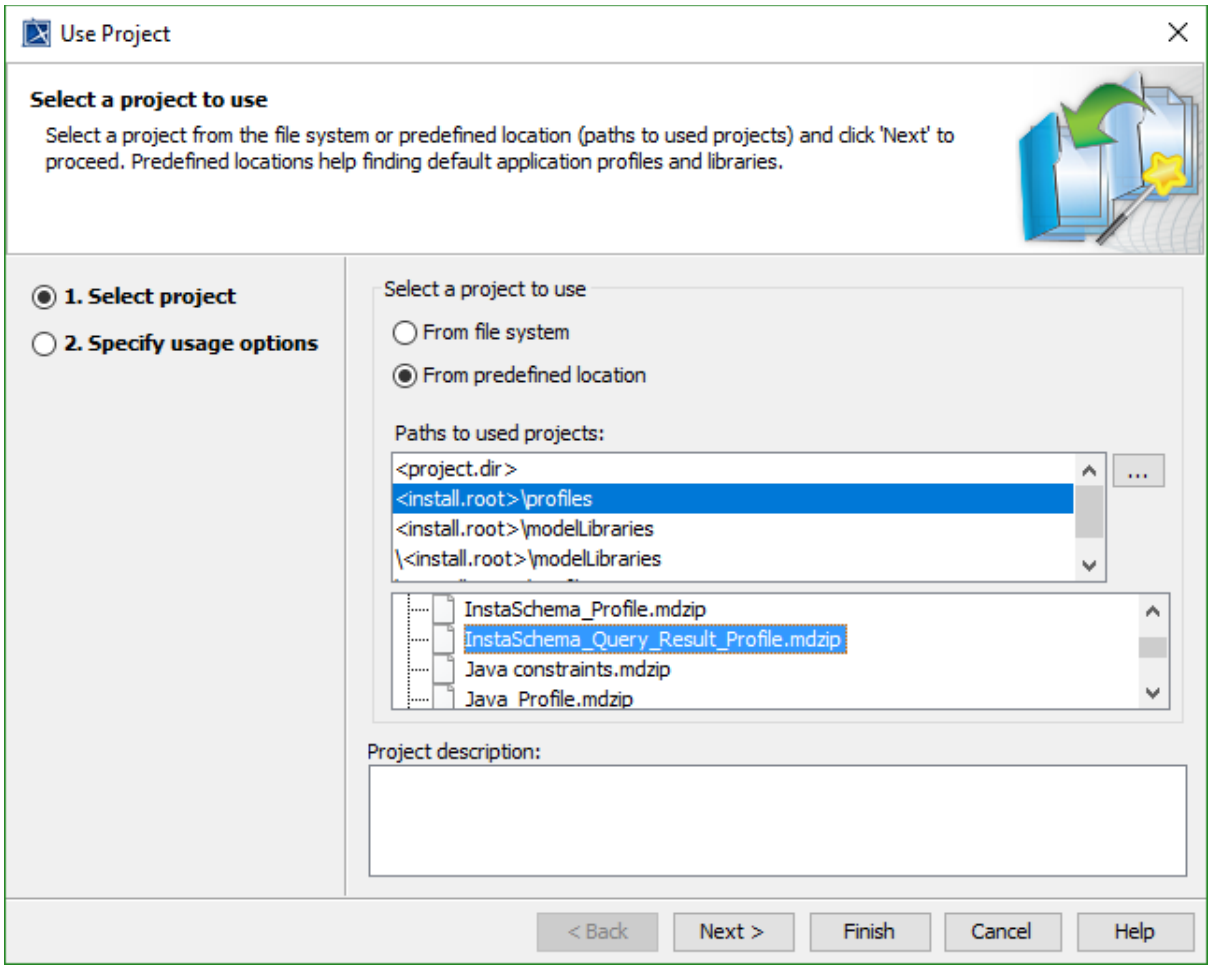
Important limitations:

- If a *QueryDefinition* defines multiple public queries the first one will be used for the validation rule.
- The annotation produced by the validation rule will be attached to the first parameter of the query.

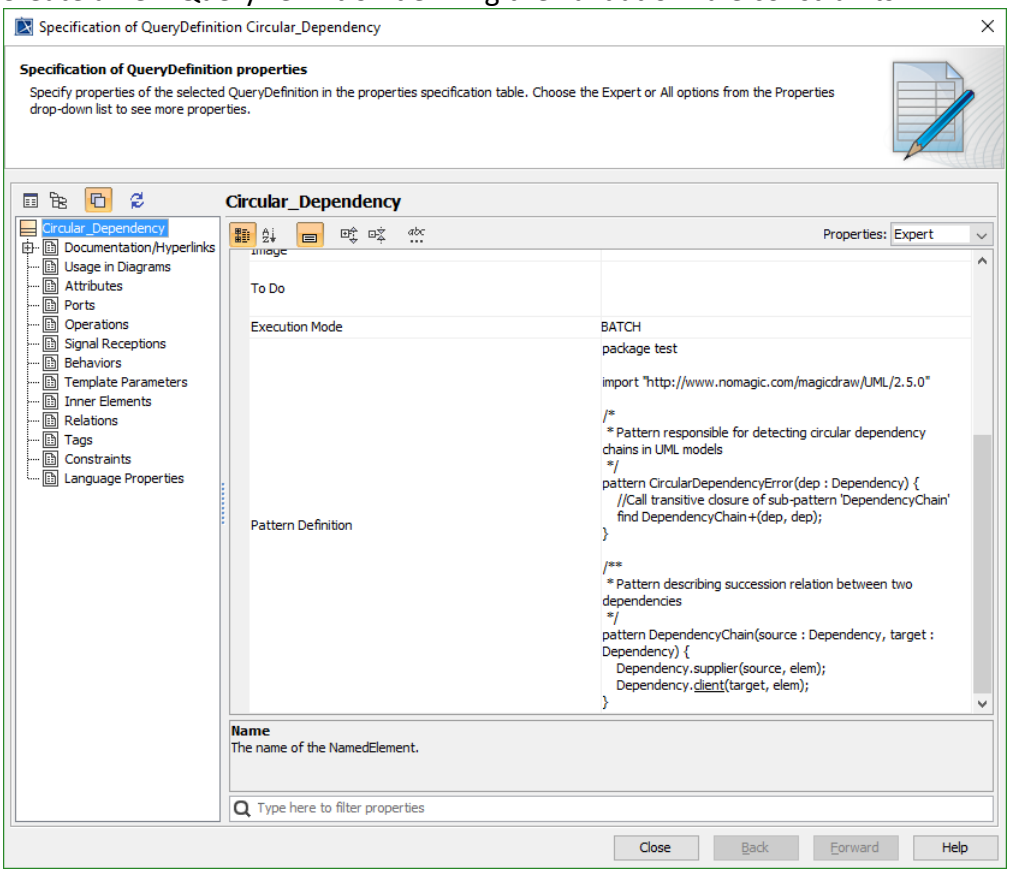
Rule Definition Example

Following section shows how to define validation rules in a MagicDraw model with InstaSchema 1.2.

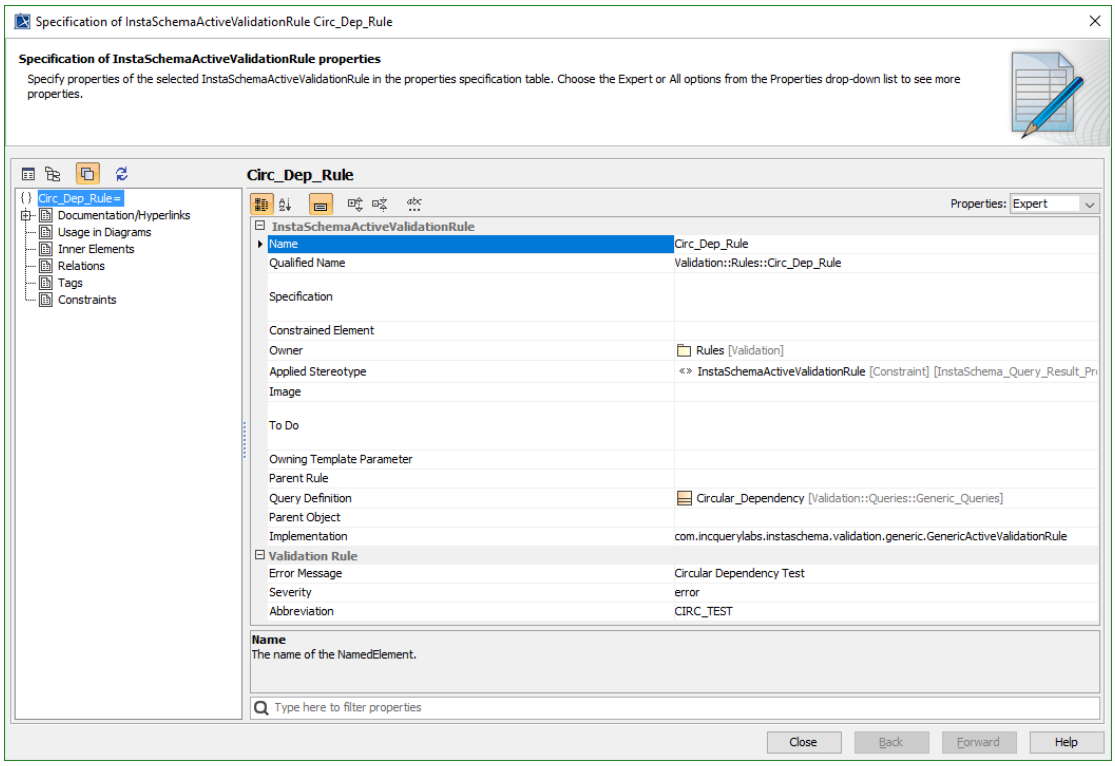
- Create a new SysML project with some content
- Import the *InstaSchema_Query_Result_Profile* via *file/Use Project/Use Local Project* (it can be found in the *profiles* folder)



- Create a new QueryDefinition defining the validation rule constraints



- Create a new MagicDraw active validation suite
- Create a new InstaSchema active validation rule
    - Make sure to create a new tagged value for the implementation property. (The actual string value is entered automatically, however the tag needs to be created)
    - Define message and severity
    - Select query definition to be used



- Add validation rule to suite
- Share the validation suite
- Reopen the project

Defining Rules via the InstaSchema Validation API (Advanced Feature)

If the active validation rules are defined in the model, the messages can only be defined as simple strings, therefore cannot contain any object-specific details. Also it is impossible to define quick fixes through these type of rules.

To solve this issue, InstaSchema 1.2 introduces a generic VIATRA-based active validation rule interface, which should be implemented and handed to the standard MD active validation.

The interface defines the following methods:

```
/*
* Generic template parameters that ensure that the implementing classes have type-safe methods
* - Match: Generated Match class of a given VIATRA Query pattern. Contains a single result
* - Specification: Query specification generated for a given VIATRA pattern
*/
public interface IQueryBasedActiveValidationRule<Match extends IPatternMatch,
    Specification extends IQuerySpecification<? extends ViatraQueryMatcher<Match>>> extends ElementValidationRuleImpl {

  //Return QuerySpecification to be used with the validation rule
  public abstract Specification getQuerySpecification();

  //Define a custom severity value
  public abstract EnumerationLiteral getSeverity(final Match match, final Constraint constr);

  //Define Annotation kind. This value should be unique for each annotated element.
  public abstract String getAnnotationKind(final Match match, final Constraint constr);

  //Define annotation message
  public abstract String getMessage(final Match match, final Constraint constr);

  //Define quick fixes for each match of the specified query
  public abstract List<? extends NMAction> getQuickFixes(final Match match, final Constraint constr);

  //Define which parameter of the query should be annotated
  public abstract Element getAnnotatedElement(final Match match);
}
```

Programmatic Rule Definition Example

Following part explains the steps necessary for defining validation rules through the API.

- Create a new MagicDrawD plugin and set up IDE according to the MagicDraw OpenAPI documentation.
- Set up a developer environment for VIATRA queries.
- Define a query that implements the desired validation rule.
- Create a new class that redefines *com.incquerylabs.instaschema.validation.api IqueryBasedActiveValidationRule.*
- Implement the methods accordingly.
- Launch MagicDraw with the new plugin on the classpath.
- Create a new validation suite and active validation rule.
    - Set the implementation: fully qualified name of the previously implemented class.
- Share suite and reload project.

Thank you for trying InstaSchema. Keep in mind however, that the tool itself is still experimental. If you encounter any issues please report them on the following forum topic.