

Modelling and Enforcing Access Control Requirements for Smart Contracts - Data Set

Master Thesis of

Jan-Philipp Töberg

at the Department of Informatics
Competence Center for Applied Security Technology (KASTEL)

Reviewer:	Prof. Dr. Ralf H. Reussner
Second reviewer:	Prof. Dr. Bernhard Beckert
Advisor:	M.Sc. Frederik Reiche
Second advisor:	M.Sc. Jonas Schiffel

13. July 2021 – 13. January 2022

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

1 Installations

1.1 Installing the Metamodel & Generator Project

To employ the metamodel and the generator, you can use the included Eclipse projects. In the following, we will explain how to install and use them:

1. Download Eclipse IDE 2021-09¹ and install the **Eclipse Modelling Tools** (see Fig. 1.1)
2. Start Eclipse and create a new workspace (or choose an already existing one)
3. Under "Help" > "Install New Software..." click "Add" and enter add the following to installation sites (each needs to be a single entry and the name can be arbitrary):
 - a) <https://updatesite.mdsd.tools/metamodel-modeling-foundations/nightly/>
 - b) <http://download.eclipse.org/modeling/mdt/ocl/updates/releases>
 - c) <https://download.eclipse.org/modeling/tmf/xttext/updates/composite/marketplace/>
 - d) <https://kit-sdq.github.io/updatesite/release/commons>

Afterwards choose "-All Available Sites-" in the drop-down menu next to the "Add..." button and install the following packages:

- MDSD.tools Modeling Foundations
- OCL All-In-One SDK
- Xtend & Xtend IDE
- SDQ Commons

If a security warning is shown, just click "Install anyway". For completeness, you can view a list of all installed software packages as well as their version for a running eclipse configuration in Figure 1.4.

4. Restart Eclipse (you will be prompted to do so)
5. If you close the welcoming screen, the "Model Explorer" View should be open. If not, you can open it under "Window" > "Show View" > "Model Explorer".
6. Import the Projects from the "MetamodelAndGeneratorProjects" folder as follows:
 - a) Right-Click in the "Model Explorer" View > "Import..."
 - b) Choose "General" > "Projects from Folder or Archive"

¹<https://www.eclipse.org/downloads/>

- c) Choose the "MetamodelAndGeneratorProjects" Directory and select all available projects as seen in Fig. 1.2. Click "Finish".
7. If there are over 200 errors in the edu.kit.kastel.sdq.soliditycodegenerator folder that state that a certain sequence cannot be resolved to a type (e.g. "Â cannot be resolved to a type."), you need to right-click on the folder > "Properties" > "Resource" (should already be selected) and change the "Text file encoding" from "Inherited from container" to "Other:" > "UTF-8". Now apply and close the properties window and after a re-build there should be no errors left.
8. In the "Model Explorer", navigate to "AccessControlMetamodel" > "model" > "AccessControlMetamodel.ecore" > "AccessControlMetamodel" and either "AccessControlSystem" or "SmartContractModel". To open the diagram in Eclipse, double click their first child element that are marked in Fig. 1.3.

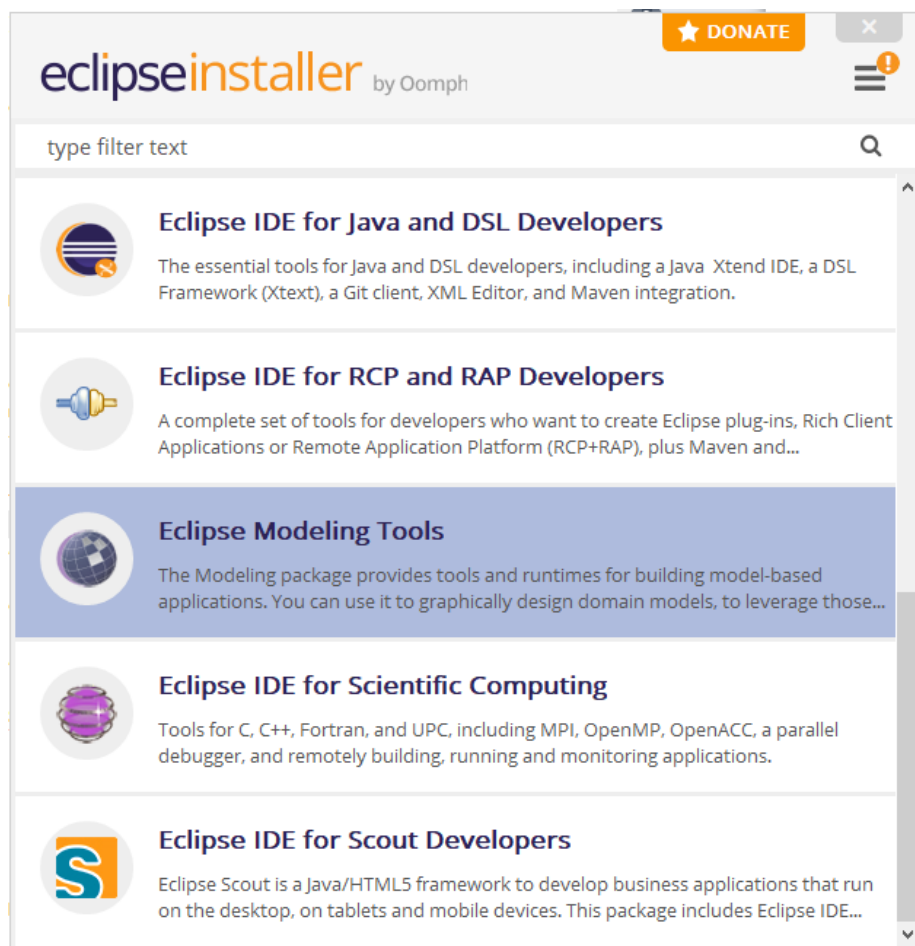


Figure 1.1: Screenshot of the Eclipse Installation that should be selected in step 1

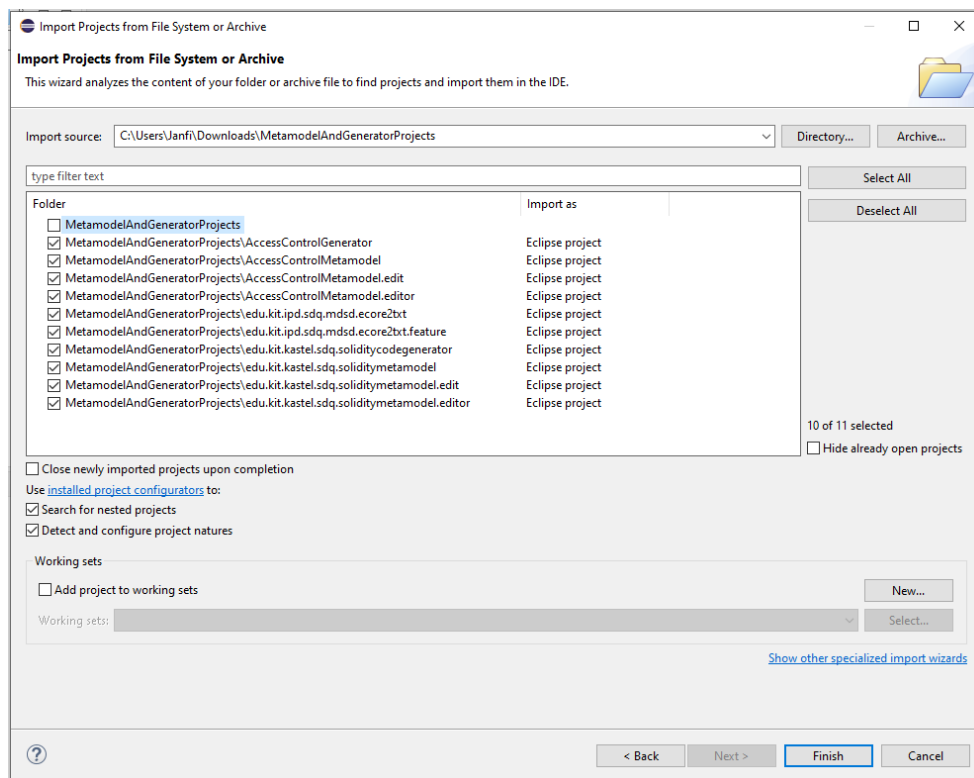


Figure 1.2: Screenshot of the Import-Window when importing the projects in step 6c.

1.2 Looking at the Examples & Using the Generator

To look at the created examples or create your own examples, as well as run the generator on these examples, the following steps need to be followed:

1. In your set-up Eclipse environment from Ch. 1.1, Right-click on either the "Access-ControlGenerator" or the "AccessControlMetamodel.editor" folder and "Run As" > "1 Eclipse application". Now, a new Eclipse window should open.
2. There should be no project open in the ProjectExplorer view, so you can click "Import projects..." and follow the same steps from step 6 in the previous chapter. This time, choose the "Examples" folder and select the only available option "Examples".
3. Now you have the different folders for each example and use cases available
4. To create a new metamodel instance, you need to make a right-click on any object in the Project Explorer and choose "New" > "Other...". In the newly opened window, navigate to "Example EMF Model Creation Wizards", where you can select either the "AccessControlSystem Model" or the "SmartContractModel Model". Press "Next".
5. Now you can choose the filename and its location. Press "Next".

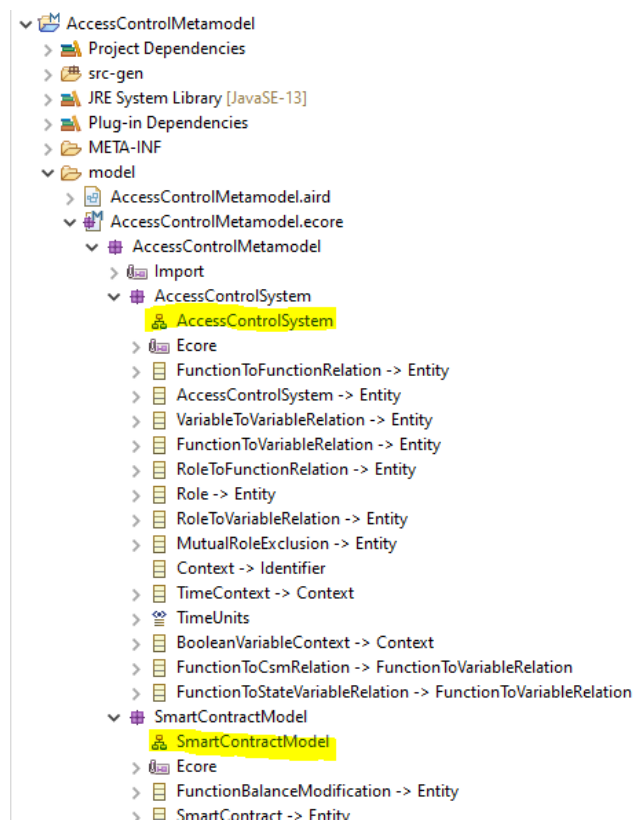


Figure 1.3: The files you need to open to see the whole diagram as explained in step 8.
















































Name	Version	Id	Provider
>  CDO Model Repository EPP	4.15.0.v20210908-1...	org.eclipse.emf.cdo.epp.feature.group	Eclipse Modeling Project
 Eclipse Commons	2.0.0	edu.kit.ipd.sdq.commons.util.eclipse.featur...	KIT SDQ
>  Eclipse Commons Developer Resources	2.0.0	edu.kit.ipd.sdq.commons.util.eclipse.featur...	KIT SDQ
 Eclipse e4 Tools	4.21.0.v20210604-0...	org.eclipse.e4.core.tools.feature.feature.gr...	Eclipse.org
 Eclipse Java Development Tools	3.18.900.v20210906...	org.eclipse.jdt.feature.group	Eclipse.org
>  Eclipse Modeling Tools	4.21.0.20210910-1200	epp.package.modeling	Eclipse Packaging Project
>  Eclipse Platform	4.21.0.v20210906-0...	org.eclipse.platform.feature.group	Eclipse.org
>  Eclipse Plug-in Development Environment	3.14.900.v20210906...	org.eclipse.pde.feature.group	Eclipse.org
>  Eclipse Project SDK	4.21.0.v20210906-0...	org.eclipse.sdk.feature.group	Eclipse.org
>  Eclipse RCP	4.21.0.v20210906-0...	org.eclipse.rcp.feature.group	Eclipse.org
>  Ecore Diagram Editor	3.3.3.20210261352	org.eclipse.emf.ecoretools.design.feature....	Eclipse Modeling Project
>  EMF Diagram Editor (SDK)	3.3.3.20210261352	org.eclipse.emf.ecoretools.sdk.feature.gro...	Eclipse Modeling Project
>  EMF - Eclipse Modeling Framework SDK	2.27.0.v20210816-1...	org.eclipse.emf.sdk.feature.group	Eclipse Modeling Project
 EMF Commons	2.0.0	edu.kit.ipd.sdq.commons.util.emf.feature....	KIT SDQ
>  EMF Commons Developer Resources	2.0.0	edu.kit.ipd.sdq.commons.util.emf.feature....	KIT SDQ
>  EMF Forms SDK	1.26.1.20210901-0930	org.eclipse.emf.ecp.emfforms.sdk.feature....	Eclipse Modeling Project
>  EMF Model Query SDK	1.12.0.201805030653	org.eclipse.emf.query.sdk.feature.group	Eclipse Modeling Project
>  EMF Model Transaction SDK	1.12.0.201805140824	org.eclipse.emf.transaction.sdk.feature.gr...	Eclipse Modeling Project
>  EMF Parsley SDK	1.13.0.v20201130-1...	org.eclipse.emf.parsley.sdk.feature.group	Eclipse Modeling Project
>  EMF Parsley SDK Developer Resources	1.13.0.v20201130-1...	org.eclipse.emf.parsley.sdk.source.feature....	Eclipse Modeling Project
>  EMF Validation Framework SDK	1.12.2.202008210805	org.eclipse.emf.validation.sdk.feature.group	Eclipse Modeling Project
>  GEF (MVC) SDK	3.11.0.201606061308	org.eclipse.gef.sdk.feature.group	Eclipse GEF
>  Git integration for Eclipse	5.13.0.20210908082...	org.eclipse.egit.feature.group	Eclipse EGit
>  Graphical Modeling Framework (GMF) Runtime SDK	1.13.1.202106221344	org.eclipse.gmf.runtime.sdk.feature.group	Eclipse Modeling Project
 Java Commons	2.0.0	edu.kit.ipd.sdq.commons.util.java.feature....	KIT SDQ
>  Java Commons Developer Resources	2.0.0	edu.kit.ipd.sdq.commons.util.java.feature....	KIT SDQ
>  Marketplace Client	1.9.1.v20210204-1408	org.eclipse.epp.mpc.feature.group	Eclipse Marketplace Client
 MDSD Profiles Commons	2.0.0	edu.kit.ipd.sdq.commons.util.mdsdprofile...	KIT SDQ
>  MDSD Profiles Commons Developer Resources	2.0.0	edu.kit.ipd.sdq.commons.util.mdsdprofile...	KIT SDQ
>  MDSD.tools Modeling Foundation Identifier UI Support Featu	1.0.0.202010011104	tools.mdsd.modelingfoundations.identifie...	
 MDSD.tools Modeling Foundations Identifier Feature	1.0.0.202010011104	tools.mdsd.modelingfoundations.identifie...	
>  Model comparison (EMF Compare) - Core - SDK	3.3.15.202107200824	org.eclipse.emf.compare.source.feature.gr...	Eclipse Modeling Project
>  Model comparison (EMF Compare) - EGit support	3.3.15.202107200824	org.eclipse.emf.compare.egit.feature.group	Eclipse Modeling Project
>  Model comparison (EMF Compare) - SDK	3.3.15.202107200824	org.eclipse.emf.compare.ide.ui.source.fea...	Eclipse Modeling Project
>  Model comparison (EMF Compare) - Sirius support (Experim	3.3.15.202107200824	org.eclipse.emf.compare.diagram.sirius.so...	Eclipse Modeling Project
>  Mylyn WikiText	3.0.38.202008172112	org.eclipse.mylyn.wikitext.feature.feature....	Eclipse Mylyn
>  OCL All-In-One SDK	6.16.0.v20210907-2...	org.eclipse.ocl.master.feature.group	Eclipse OCL
>  OCL All-In-One SDK Developer Resources	6.16.0.v20210907-2...	org.eclipse.ocl.master.source.feature.group	Eclipse OCL
>  OCL Classic SDK: Ecore/UML Parsers,Evaluator,Edit	5.16.0.v20210907-2...	org.eclipse.ocl.all.sdk.feature.group	Eclipse OCL
>  Oomph Setup	1.23.0.v20211008-0...	org.eclipse.oomph.setup.feature.group	Eclipse Oomph Project
 PCM Commons	2.0.0	edu.kit.ipd.sdq.commons.util.pcm.feature...	KIT SDQ
>  PCM Commons Developer Resources	2.0.0	edu.kit.ipd.sdq.commons.util.pcm.feature...	KIT SDQ
 Tip of the Day UI Feature	0.2.1500.v20210706...	org.eclipse.tips.feature.feature.group	Eclipse.org
>  UML2 Extender SDK	5.5.2.v20210228-1829	org.eclipse.uml2.sdk.feature.group	Eclipse Modeling Project
>  XSD - XML Schema Definition SDK	2.26.0.v20210617-1...	org.eclipse.xsd.sdk.feature.group	Eclipse Modeling Project
>  Xtend	2.2.0.v201605260315	org.eclipse.xtend.feature.group	Eclipse Modeling Project
>  Xtend IDE	2.25.0.v20210301-1...	org.eclipse.xtend.sdk.feature.group	Eclipse Xtend

Figure 1.4: All installed software versions in an Eclipse environment where the metamodel and the generator work without any problems.

6. Now you can choose the "Model Object" and the "XML Encoding". For the "Model Object" you need to choose either "Access Control System" or "Smart Contract". The Encoding does not need to be changed ("UTF-8"). Finish the creation.
7. Now the newly created file is opened and new elements can be added with a right-click. Their properties can be looked at and changed in the "Properties" view.
8. If you want to generate Solidity code based on the models, you need to select at least one ".smartcontractmodel" and one ".accesscontrolsystem" file and right-click. Now the generator can be started by selecting "Access Control Generator" > "Generate Smart Contract with Access Control". Now a soundness check is started before the generation. If that check finds any problems, it communicates them back to the developer. In any case, a new folder called "gen" is created. The generator puts the created smart contracts there or a .log file if the soundness check fails.

1.3 Installing the Slither printer

To use slither and our implemented printer on a Solidity smart contract, we use Linux. To setup all necessary tools and packages, you need to follow these steps:

1. Install Python 3.6+ with pip (no further instructions are given since it is often pre-installed or the installation depends on the linux system you are using)
2. Install *solc*, the Solidity compiler by following these instructions
3. Install Slither using the following command:

```
$ git clone https://github.com/crytic/slither.git
$ cd slither
$ python3 setup.py install
```

If for some reason that does not work, you can see some alternatives in Slithers documentation²

4. Copy the "influence_and_calls_plugin" folder provided with these instructions to the same folder where slither was installed
5. Navigate inside the "influence_and_calls_plugin" folder and open a new terminal
6. Enter the following command to install the custom printer:

```
$ python3 setup.py develop
```

7. If no errors occurred during the installation, the printer is now successfully installed. To test it, you can enter the following command in any folder that contains the Solidity file <Filename> that you want to analyze:

```
$ slither <Filename>.sol --print influence-and-calls
```

²<https://github.com/crytic/slither#how-to-install>

2 Metamodel

Our created metamodel consists of two different packages that are separated to enhance the separation of concerns. The first package can be seen in Fig. 2.1 and covers all aspects of our model needed to describe a basic Solidity contract. This includes the functions, state variables and the needed data types as well as the possible balance modifications. This package uses elements from the SolidityMetaModel¹.

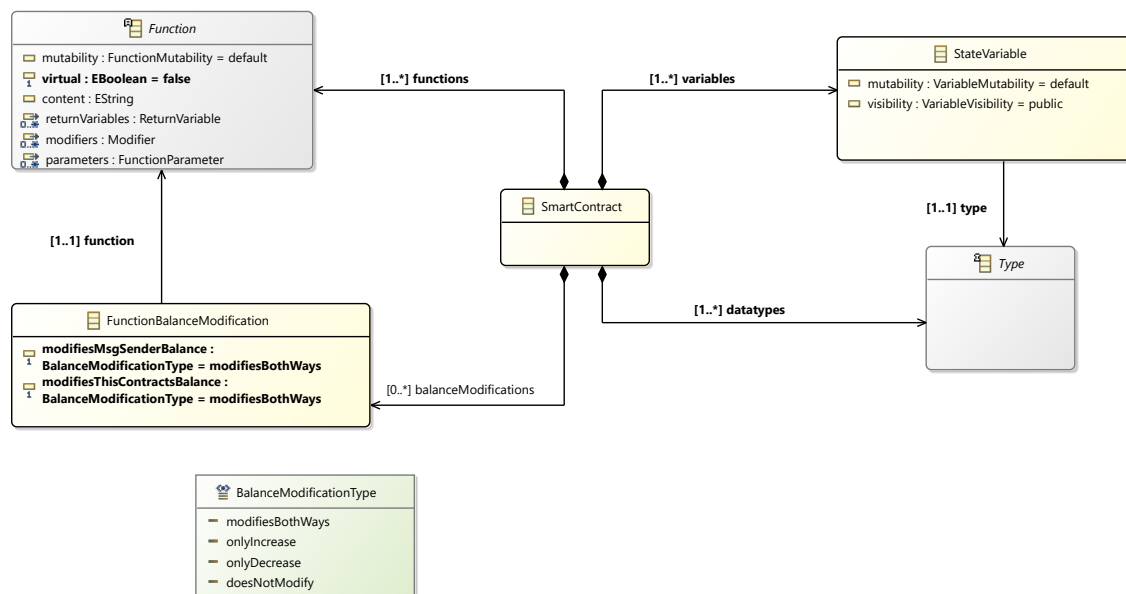


Figure 2.1: First half of our newly created metamodel to describe RBAC requirements for smart contracts. This package contains all aspects that rely on smart contract / Solidity specific aspects.

The second package covers the remaining aspects of our approach by describing all elements needed to model RBAC policies for the smart contract described with the first package. It can be seen in Fig. 2.2 and covers roles as well as all relation elements that relate functions, state variables and roles with each other. Additionally, a context can be provided for certain kinds of relations.

¹<https://github.com/KASTEL-CSSDA/SolidityMetaModel>

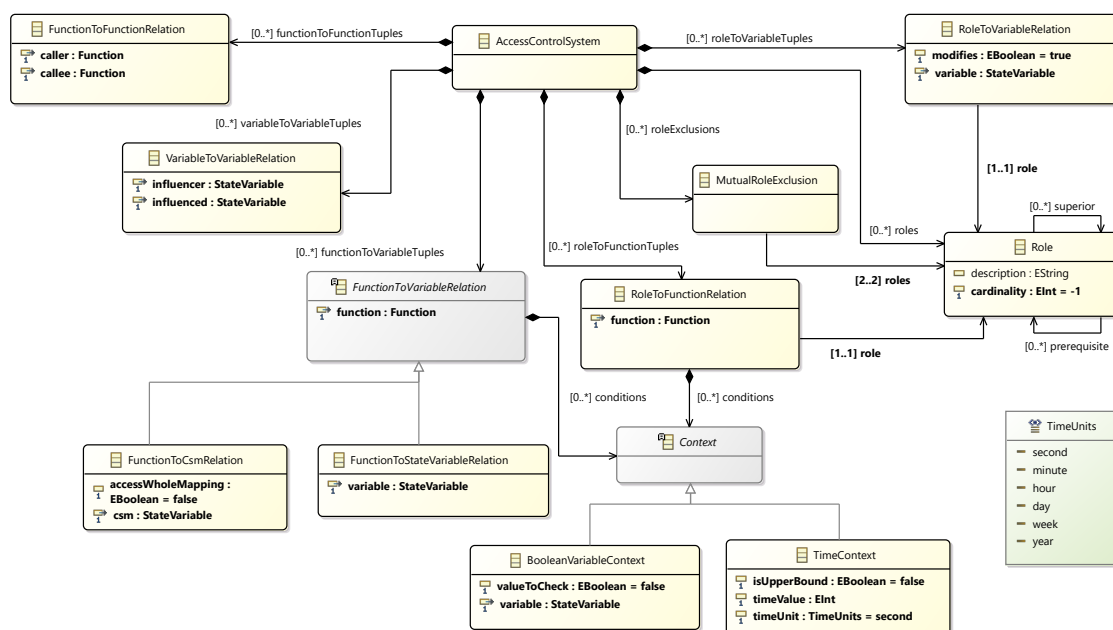


Figure 2.2: Second half of our newly created metamodel to describe RBAC requirements for smart contracts. This package contains all aspects that are needed to model the RBAC policies.

2.1 Project Overview

This guideline comes with the relevant projects ready to be imported into an Eclipse project as explained in step 6 of Ch. 1.1. However, not all of them need to be reviewed since most of them are only needed to correctly run the projects:

- **AccessControlGenerator** - This folder contains the implemented code generator that will be explained in more detail in Ch. 3. However, only the packages contained in the "src" folder are relevant for the review since the files in the "xtend-gen" folder are derived automatically from the ones in the "src" folder.
- **AccessControlMetamodel** - This folder contains the metamodel and the generated Java source code. To access the models, follow step 8 from the instruction in Ch. 1.1. This project also contains the OCL constraints that are topic of the next chapter. However, to access them, you need to open the ecore file (see Fig. 1.3) with a double-click. In the now opened tree hierarchy, you can navigate to the model elements that contain OCL constraints. The constraints definition can be found under [Element] > "Pivot". Here, all constraints are saved as a key-value-pair, with the key giving the constraint a name and the value containing the actual constraint (see Fig. 2.3).
- **AccessControlMetamodel.edit & .editor** - Both folders are generated automatically based on the model and are needed to create model instances. They do not need to be reviewed.

- **edu.kit.ipd.sdq.mdsd.ecore2txt & .feature** - Foundation for the generator implementation, taken from the GitHub repository². They do not need to be reviewed.
- **edu.kit.kastel.sdq.soliditycodegenerator** - Solidity smart contract generator for the SolidityMetaModel. The generator works as a foundation for our generator, is taken from GitHub³ and does not need to be reviewed.
- **edu.kit.kastel.sdq.soliditymetamodel** - The SolidityMetaModel is an EMF meta-model used to describe Solidity smart contracts that works as a foundation for our metamodel. This is taken from its GitHub repo⁴ and does not need to be reviewed.
- **edu.kit.kastel.sdq.soliditymetamodel.edit & .editor** - Both are automatically generated and do not need to be reviewed.

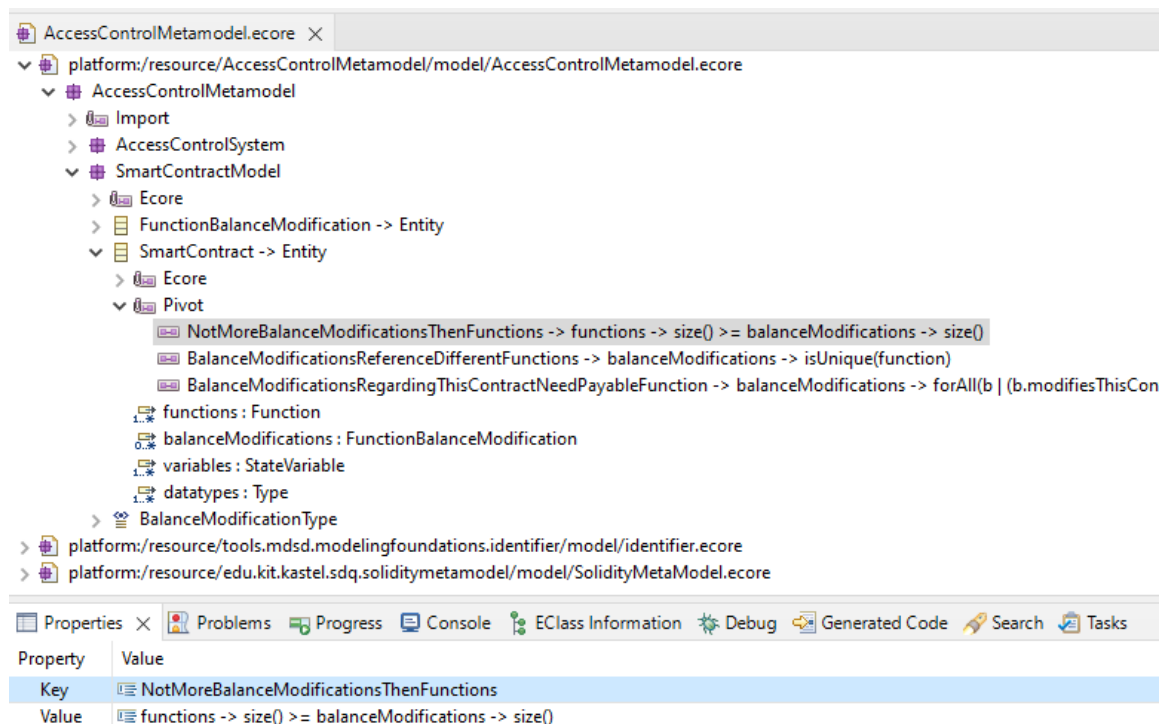


Figure 2.3: How to access the OCL constraints in our metamodel.

²<https://github.com/kit-sdq/Ecore2Txt>

³<https://github.com/KASTEL-CSSDA/SolidityCodeGenerator>

⁴<https://github.com/KASTEL-CSSDA/SolidityMetaModel>

3 XTend Generator

The generator, which uses XTend¹ to create the Solidity smart contracts based on the metamodel instances, is based on the Ecore2Txt project² and the SolidityCodeGenerator³. The latter is the generator developed to create Solidity smart contracts based on the SolidityMetaModel, which is referenced by our AccessControlMetamodel.

In general our generator can be found in the "AccessControlGenerator" project folder, after it was imported into Eclipse (see Ch. 1.1 for the explanation). The Xtend classes that contain the generation logic can be found in the "src" folder, the files in the "xtend-gen" folder can be ignored for the review since they are automatically generated from these XTend classes. All in all, the classes and methods we implemented are all documented inside the code, so we provide only basic explanations in this chapter. All packages that we explain in the following subchapters are part of the "src" folder.

3.1 edu.kit.kastel.sdq.accesscontrolgenerator.generators & .handlers

These two packages contain only three classes that are mainly derived from the Ecore2Txt project⁴ and are used as the starting point of the generator. Especially the *AccessControlGeneratorHandler* class is used to delegate the command that is issued when the generator is selected in Eclipse's context menu (see step 8 in Chapter 1.2). This class refers the call to the *AccessControlGenerator* with the help of the *AccessControlGeneratorModule*. In the *AccessControlGenerator*, the input files are preprocessed (using the soundness check explained in more detail in Ch. 3.3) before the file contents are generated using the different generator classes explained below.

3.1.1 accesscontrolsystem Subpackage

This subpackage contains all generators and supporting classes necessary to create all access control related aspects inside the generated smart contracts as well as the additional access control contract:

- **AccessControlConstants:** This class contains all constant string values that are needed throughout the generation and are relevant for access control. This includes,

¹<https://www.eclipse.org/xtend/>

²<https://github.com/kit-sdq/Ecore2Txt>

³<https://github.com/KASTEL-CSSDA/SolidityCodeGenerator>

⁴https://sdqweb.ipd.kit.edu/wiki/Generating_code_with_Xtend_and_Xtext_triggered_from_the_Eclipse_context_menu

for example, the access control contract name or the constant parts of the solc-verify annotations.

- **AccessControlContractGenerator:** This class fills the *SolidityContractGenerationTemplate* (see Ch. 3.2) with the contents that are relevant for the additionally created access control contract.
- **AccessControlSupportFunctionality:** This class provides additional functionality that is used throughout the generation, like removing duplicate elements from lists and checking if the additionally created access control contract needs to be imported.
- **AccessControlValidator:** With this class, the soundness check and the validation of OCL constraints is done. The aspects that are considered during this validation are explained in more detail in Ch. 3.3.
- **AnnotationGenerator:** This class is used to generate the annotations for the different functions using the solc-verify syntax to specify modifications to variables by functions based on the modelled policies.
- **ModifierGenerator:** This class is used to generate the access control related modifiers for the smart contract. This includes the role checks as well as the conditions that can be modelled.
- **ModifierRoleAndConditionsHelper:** This helper class is used to handle the connections between roles and the conditions defined for the functions they can access. This is used in the *ModifierGenerator* to collect all necessary objects in one class.
- **TransitiveClosureCalculator:** This class is given an *AccessControlSystem* from the model and calculates the needed transitive closures. These include the transitive closure of variable to variable influence as well as the function calls.

3.1.2 smartcontract Subpackage

This subpackage contains all generators and supporting classes necessary to create the basic smart contracts without their access control related aspects:

- **SolidityConstants:** This class contains all constant string values that are needed throughout the generation and are connected mainly to the Solidity smart contract. This includes, for example, the file extension and prefix for the target folder. This class is copied from the *SolidityCodeGenerator*.
- **SolidityContractGenerator:** This class assembles the different parts of the modelled smart contract and generates a Solidity file for it by filling the *SolidityContractGenerationTemplate* (see Ch. 3.2). To do so, it also uses access control related classes to get the modifiers or the annotations.

- **SolidityFunctionGenerator:** This class is used to create a Solidity function based on a function from the model. To do so, it fills the *SolidityFunctionGenerationTemplate* (see Ch. 3.2) with the fitting values.
- **SolidityNaming:** This class is used to convert all strings related to naming and Solidity keywords to the right typing. This class is copied from the SolidityCodeGenerator.

3.2 edu.kit.kastel.sdq.accesscontrolgenerator.generators. templates

This package contains the templates used by the generator classes. They provide the general structure of a document or part of the document, whereas each structural part is filled by at least one generator:

- **M2T-Generator:** Interface providing the generation functionality
- **SolidityContractGenerationTemplate:** Filling the generate() method of the *M2T-Generator* interface with different methods for the structural elements of a Solidity smart contract. This class is implemented by the contract generators, which fill these methods with the needed content.
- **SolidityFunctionGenerationTemplate:** Filling the generate() method of the *M2T-Generator* interface with different methods for the structural elements of a Solidity function. This class is implemented by the *SolidityFunctionGenerator*, which fills these methods with the needed content.

3.3 Soundness Check

To reason about the soundness of the metamodel instances, we use the *AccessControlValidator* class. In addition to checking the defined OCL constraints, it checks the fulfillment of the RBAC equations seen below. If any violations are found, they are communicated back to the developer and the generator stops the generation by creating a log-file, allowing only instances without any violations to result in functioning Solidity smart contracts.

Definitions:

\mathbb{R} = Roles, \mathbb{F} = Functions, \mathbb{S} = State Variables

$ARF : \mathbb{R} \times \mathbb{F}$ (**A**ccess**R**ole**F**unction - Role may call function)

$MRS : \mathbb{R} \times \mathbb{S}$ (**M**odification**R**ole**S**tateVariable - Role may modify state variable)

$MFS : \mathbb{F} \times \mathbb{S}$ (**M**odification**F**unction**S**tateVariable - Function may modify state variable)

$IRS : \mathbb{R} \times \mathbb{S}$ (**I**nfluence**R**ole**S**tateVariable - Role may influence state variable)

$ISS : \mathbb{S} \times \mathbb{S}$ (**I**nfluence**S**tateVariable**S**tateVariable - First state variable may influence the second state variable)

$CFF : \mathbb{F} \times \mathbb{F}$ (**C**all**F**unction**F**unction - First function may call the second function)

$MER : \mathbb{R} \times \mathbb{R}$ (**M**utual**E**xclusion**O**f**R**oles - The two roles are mutually exclusive)

If a role may access a function, it also needs access to all functions called by it:

$$\forall r \in \mathbb{R}, f, f_c \in \mathbb{F} : (r, f) \in ARF \wedge (f, f_c) \in CFF \rightarrow (r, f_c) \in ARF \quad (3.1)$$

If a role may modify a state variable, it needs to have access to at least one function that can modify the variable:

$$\forall r \in \mathbb{R}, s \in \mathbb{S} : (r, s) \in MRS \rightarrow \exists f \in \mathbb{F} : (r, f) \in ARF \wedge (f, s) \in MFS \quad (3.2)$$

If a role may modify a state variable, but may not influence a second state variable, the first one may not influence the second one:

$$\forall r \in \mathbb{R}, s_1, s_2 \in \mathbb{S} : ((r, s_1) \in MRS \wedge (r, s_2) \notin IRS) \rightarrow (s_1, s_2) \notin ISS \quad (3.3)$$

If a role has access to a function, but may not influence a state variable, the function should not be able to influence the state variable:

$$\forall f \in \mathbb{F}, s \in \mathbb{S}, r \in \mathbb{R} : ((r, f) \in ARF \wedge (r, s) \notin IRS) \rightarrow \neg \text{doesInfluence}(f, s) \quad (3.4)$$

Function f does not influence state variable s if it has no modifying access to the variable itself, does not modify other state variables that may influence it and if no function that f calls influences s :

$$\neg \text{doesInfluence}(f, s) \leftrightarrow ((f, s) \notin MFS \wedge (\forall s_i \in \mathbb{S} : (f, s_i) \notin MFS \vee (s_i, s) \notin ISS) \wedge (\forall f_c \in \mathbb{F} : (f, f_c) \notin CFF \vee \neg \text{doesInfluence}(f_c, s))) \quad (3.5)$$

4 Slither Printer

To collect the necessary data about the implemented smart contracts to reason about the correct enforcement of access control policies, we extend the *Slither* framework¹ by implementing a custom printer that calculates direct, indirect and transitive influence between variables as well as the transitive closure for function calls. The printer can be found in the "influence_and_calls_plugin" folder provided with this instruction and it contains the following files:

- `SlitherCustomPrinter.md` - README file summarizing the general development approach. Also contains relevant links and hints to better understand the API and its implementation.
- `setup.py` - Used to install the printer as mentioned in step 7 in Chapter 1.3. Also it contains meta-information about the printer like the author and the version. Its structure is normal for slither printer as explained in the example².
- `__init__.py` - Also based on Slithers custom printer structure. Summarizes the printers and detectors that should be added with this plugin.
- **`influence_and_calls.py`** - This file contains our printer and all its functionality, so it should be reviewed. This is partly based on the *data-dependency* printer³, which is a standard part of the framework. All necessary explanations about its functionality are given in the comments.

It needs to be remarked that the printer provides an overapproximation of the influence and function call relations since it "blindly" calculates the transitive closure at the end.

¹<https://github.com/crytic/slither>

²https://github.com/crytic/slither/tree/master/plugin_example

³https://github.com/crytic/slither/blob/master/slither/printers/summary/data_dependency.py