HUSACCT

Software Architecture Reconstruction and Compliance Checking Tool

USER Manual

November, 2014

INTRODUCTION

HUSACCT provides support for software architecture compliance checking (SACC) of Java and C# systems. SACC monitors the conformance of the implemented architecture (in the source code) to the intended software architecture (in the system's design).
The prominent feature of HUSACCT is the support of semantically rich modular architectures (SRMAs)[10], which are composed of modules of different types (like software subsystems, layers and components) and rules of different types. To perform an SACC, an intended software architecture is defined first. Next, HUSACCT checks the compliance to these rules, based on static analysis of the source code, and it reports infringements.
Apart from ACC-support, HUSACCT provides support for architecture reconstruction as well: the static structure of the software may be visualized, browsed, and reported. Based on the obtained insight, an intended modular architecture may be defined.

To download the build or watch an introduction video, visit: http://husacct.github.io/HUSACCT/

Free-to-Use Open-Source
You can make use of HUSACCT free of charge.
This program (HUSACCT) is free software under the terms of the GNU Affero General Public License as published by the Free Software Foundation, either version 3 of theLicense, or (at your option) any later version. You can redistribute the software and/or modify it for your own use, but you are not allowed to include the software, parts of the software or documentation, in other products (for commercial or non-commercial use).
This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
See the GNU Affero General Public License for more details: http://www.gnu.org/licenses/ .

HUSACCT Development
HUSACCT means: HU Software Architecture Compliance Checking Tool, where HU stands for: HU University of Applied Sciences Utrecht.The HUSACCT project is conducted at the Institute for ICT, located in Utrecht, The Netherlands. Students of the specialization "Advanced Software Enhineering" have participated actively during the spring semesters of 2011-2013.
During the Spring semester of 2011, four teams of students developed the first prototypes.
During the spring semesters of 2012 and 2013, six teams, with four to five students per team, worked concurrently on the tool.
In December 2012 version 1.0 was released and in September 2013 version 2.0.
Since then, the development of HUSACCT has been going on.

TABLE OF CONTENTS

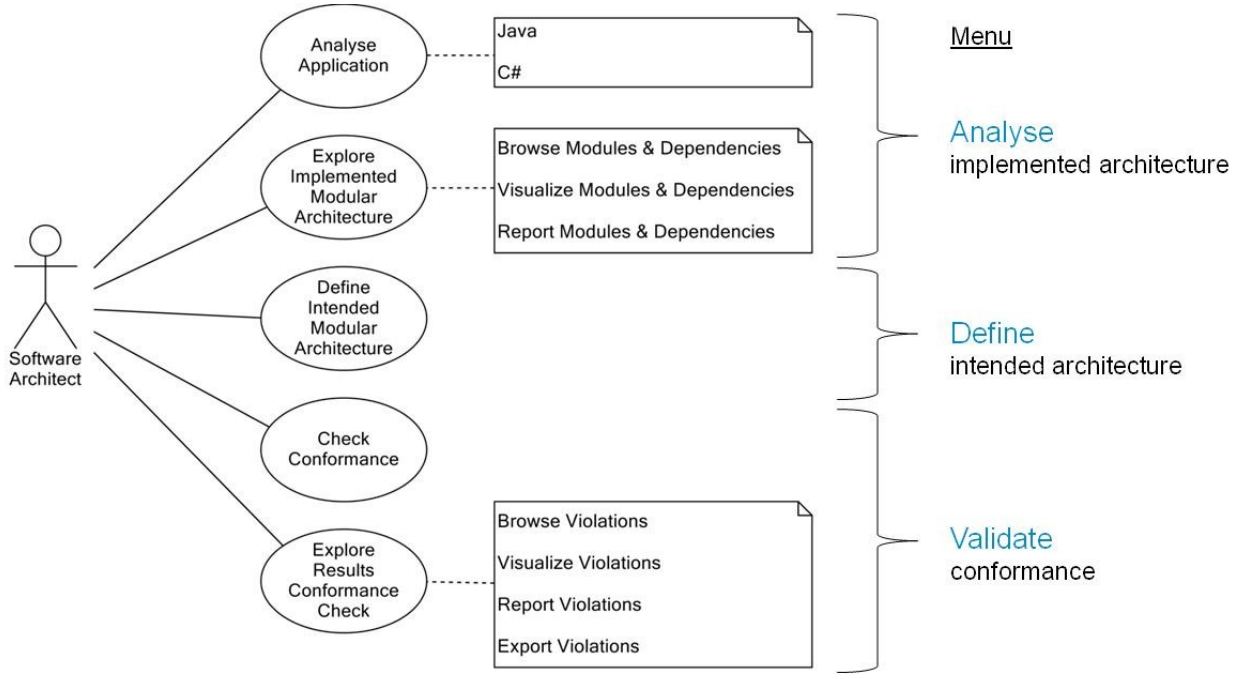
1 GETTING STARTED

1.1 DOWNLOAD AND RUN HUSACCT

1) Visit: http://husacct.github.io/HUSACCT/
   At this site you can watch an introduction video, access the documentation and download the latest release of HUSACCT. Select “Download HUSACCT_x.x JAR File” and save the jar in a directory.
2) The tool starts when you open or double click the jar-file (if your local Java-settings are ok).
   Note: HUSACCT requires Java 1.7.
3) If not (and you are running Windows), create a shortcut and edit Target to:
   java -jar <pathToHUSACCT>HUSACCT_x.x.jar
   For example: java –jar C:\Tools\HUSACCT\HUSACCT_3.2.jar,
   or: "C:\Program Files (x86)\Java\jre7\bin\java.exe" -jar C:\Tools\HUSACCT\HUSACCT_3.2.jar
4) Alternatively, start a Command prompt, cd to the directory with the HUSACCT jar and start the tool from the command line, for example: C:\Tools>java -jar HUSACCT_3.2.jar.
   If needed, check the Java version with command: java –version.

1.2 OVERVIEW OF HUSACCT’S FUNCTIONALITY

HUSACCT (HU Software Architecture Compliance Checking Tool) is a tool that provides support to analyze implemented architectures, define intended architectures, and execute conformance checks. Browsers, diagrams and reports are available to study the decomposition style, uses style, generalization style and layered style [2] of intended architectures and implemented architectures[13].
The diagram below shows an overview of the provided functionality, accessible via the menu options.



1.2.1 OVERVIEW OF THE MENU OPTIONS

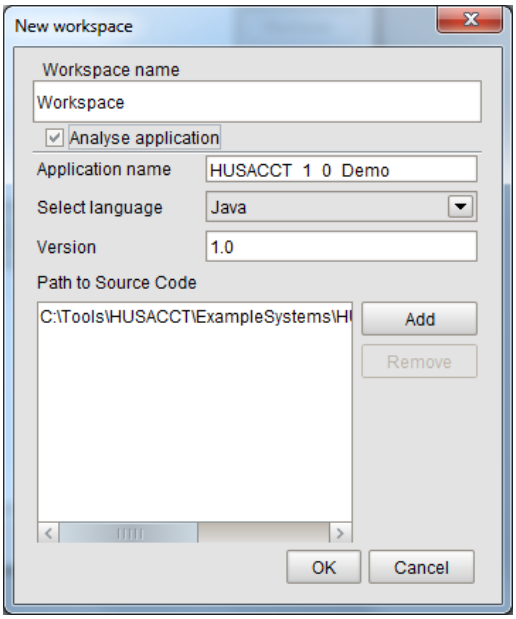
The remainder of this user manual describes and illustrates the functionality per menu option. The table below provides an overview of the options per menu.

Menu	Option
File	New Workspace
	Open Workspace
	Save Workspace
	Close Workspace
	Exit
Define intended architecture	Define intended architecture
	Intended architecture diagram
	Import architecture
	Export architecture
Analyse implemented architecture	Application properties
	Analyse application
	Analysed application overview
	Implemented architecture diagram
	Analysis history
	Export dependencies
Validate conformance	Validate now
	Violation report
Tools	Options
Help	About HUSACCT
	Documentation

1.2.2 TOUR: OVERVIEW OF THE WORK PROCESS

Follow the steps below to introduce yourself to the main functionality. Make use of the detailed instructions per menu option in the following chapters.

1) Select a software system to work on, acquire the source code, and (if available) the intended modular architecture and related architectural rules and guidelines.
2) Create a Workspace
   Menu: File => New workspace.
   Mark "Analyse Application", enter the required data, and click on the OK-button.

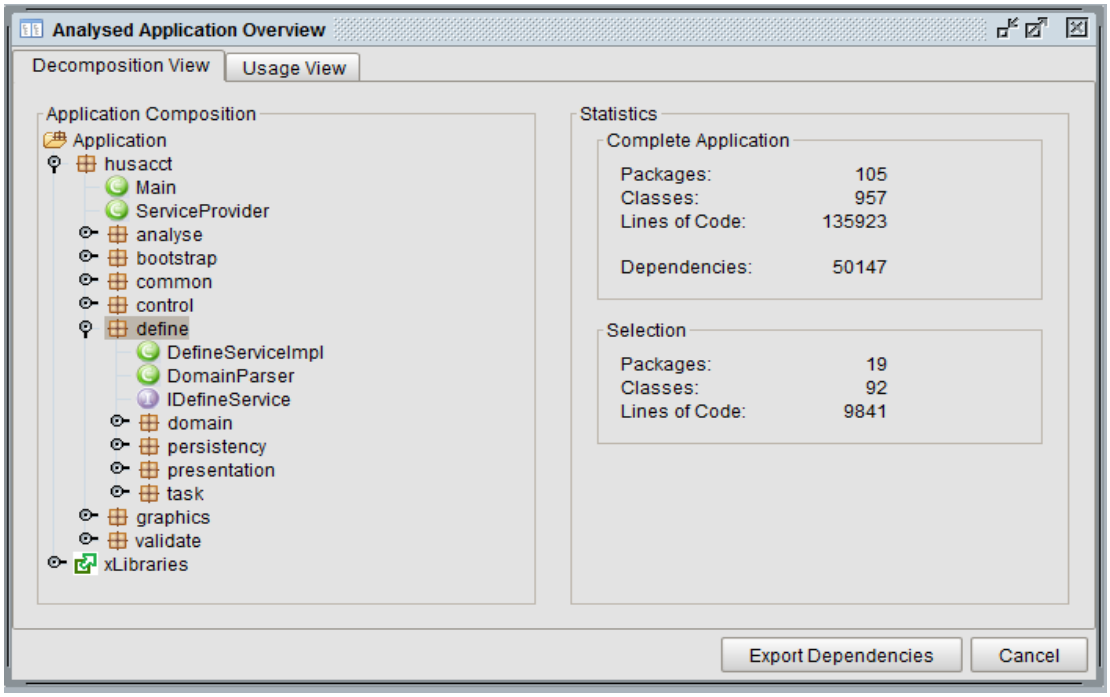


3) Study the implemented architecture

a. Analysed application overview

Menu: Analyse implemented architecture => Analysed application overview

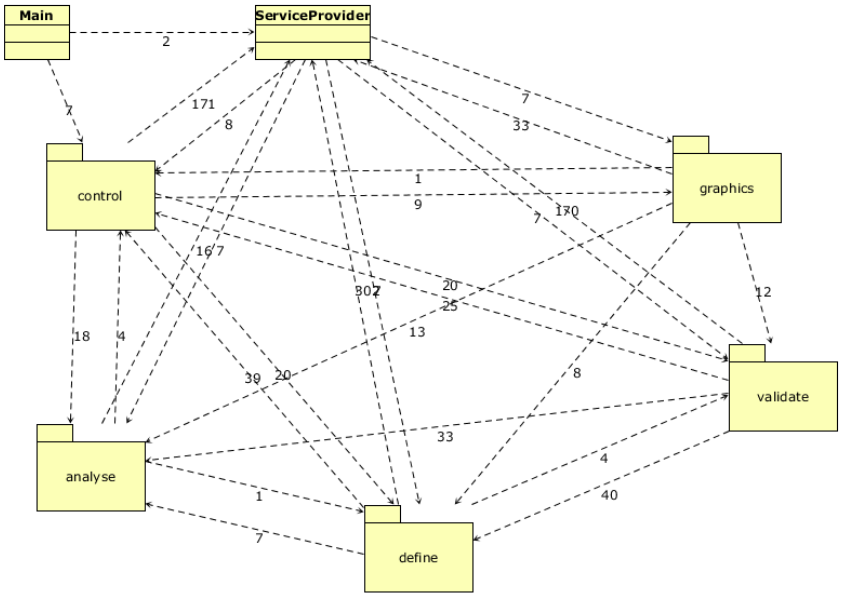
Study the Decomposition view, the related statistics, and the Usage view.



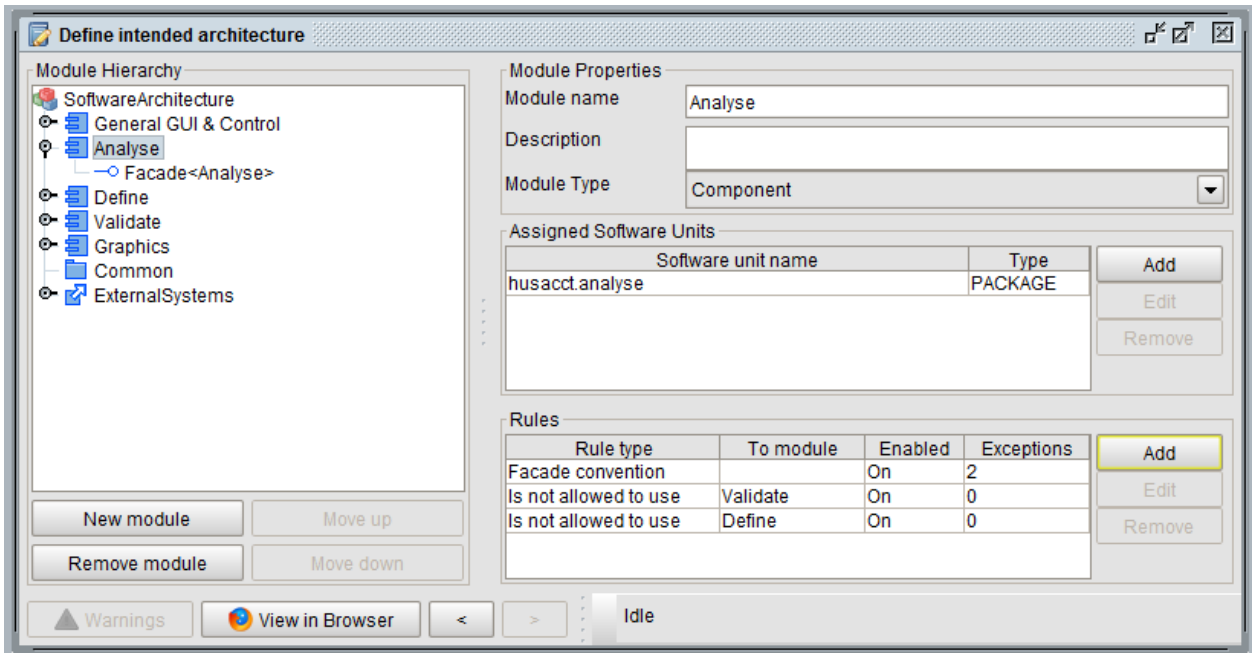
b. Implemented architecture diagram

Menu: Analyse implemented architecture => Implemented architecture diagram.

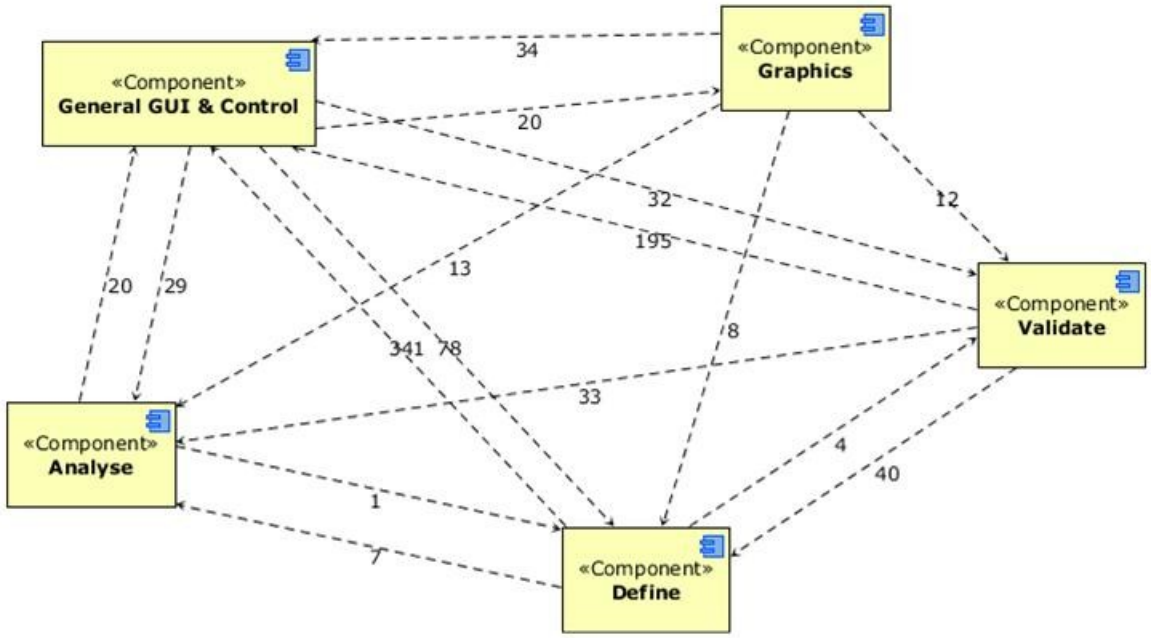
Make use of the zoom options.



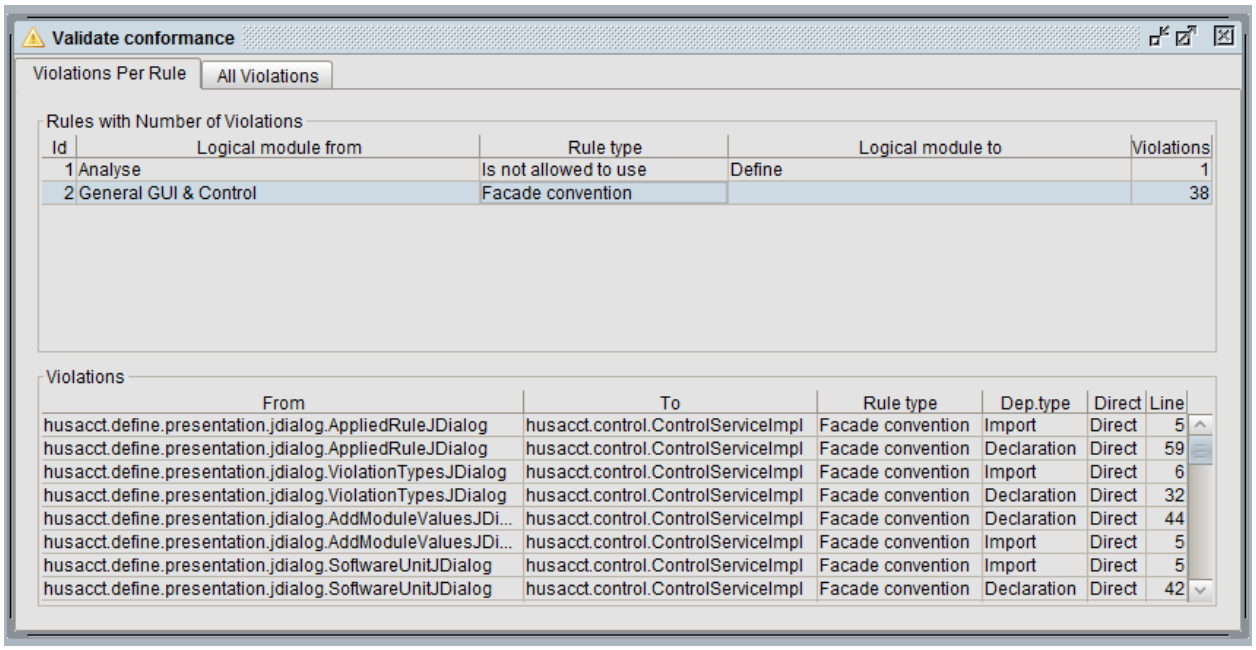
4) Define the intended architecture
   a. Create modules, add rules, and assign implemented software units to the modules.
      Menu: Define intended architecture => Define intended architecture



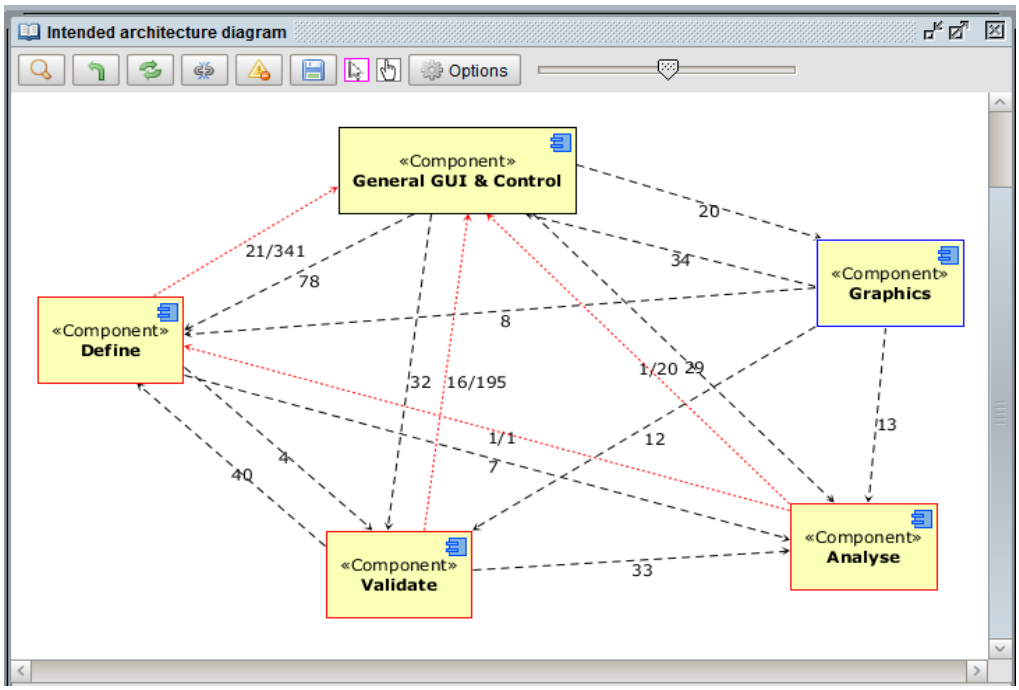
   b. Intended architecture diagram
      Menu: Define intended architecture => Intended architecture diagram.
      Make use of the options to zoom in, hide modules, or save the diagram as a picture.



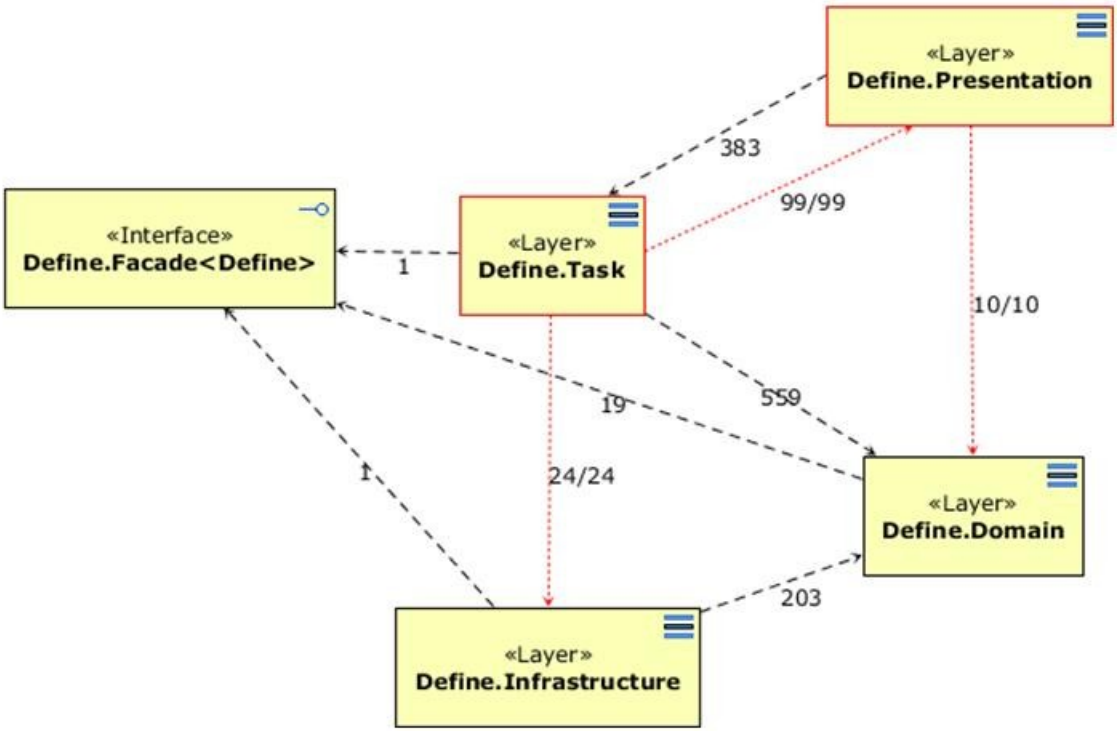
5) Check the conformance of the Implemented architecture tot the Intended architecture
   a. Check the conformance
      Menu: Validate conformance => Validate now.



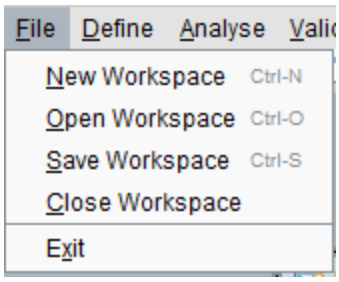
   b. Intended architecture diagram with violations
      Menu: Define intended architecture => Intended architecture diagram.
      Activate the options ‘Show violations’.



a. Implemented architecture diagram with violations
   Menu: Analyse implemented architecture => Implemented architecture diagram.
   Activate the options 'Show violations'.

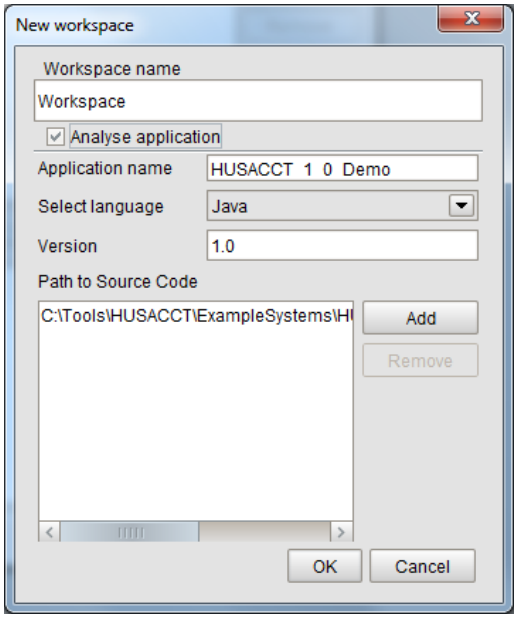


2 MENU: FILE



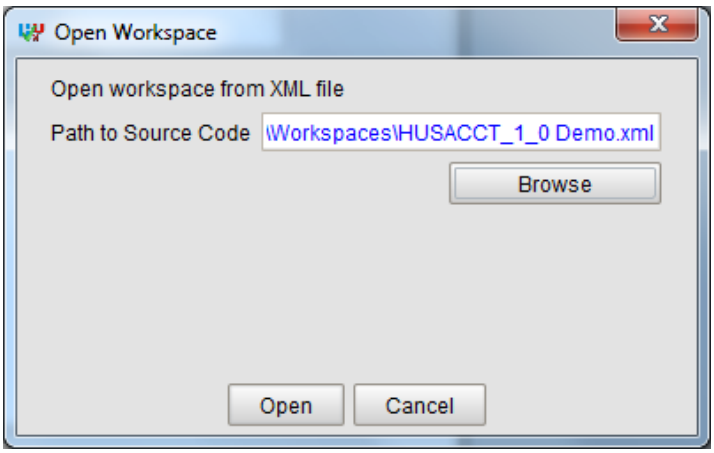
This menu allows you to manage your current workspace.
A workspace within HUSACCT contains all the information needed to analyse a target software system, study its implemented architecture, define its intended architecture and perform a compliance check. The workspace data may be stored in a file, which allows you to continue later on. Without a workspace, you cannot start working.

2.1 NEW WORKSPACE



6) To create a workspace,
   select File => New workspace.
7) Enter a name.
8) Select OK, if you want to start defining a new intended architecture.
9) Select "Analyse Application", if you first want to analyse the source code. If so, continue with the following steps.
10) Select the programming language.
11) Enter the version number of your application (not the Java version number).
12) Click on "Add" and select the directory where the *source code* is located. If needed, you can add several paths, or remove (old) paths.
13) Click "OK" and HUSACCT will start analysing the implemented application.
    Thereafter, the implemented architecture may be studied (menu 'Analyse implemented architecture').
    Furthermore

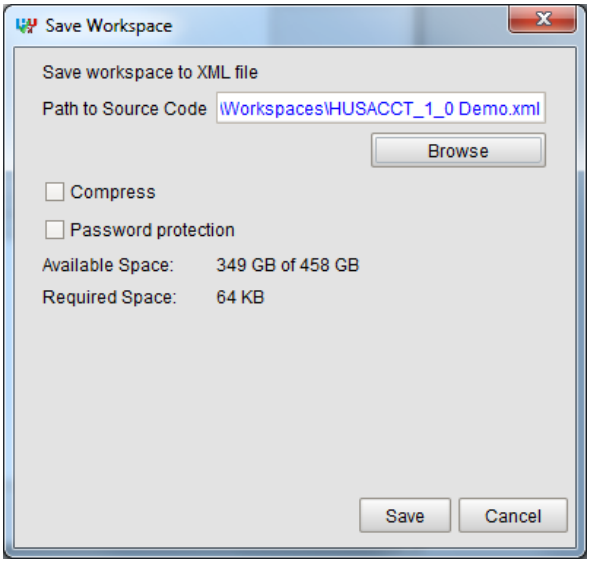
2.2 OPEN WORKSPACE



14) To open a workspace, select File => Open workspace
15) Select the file you want to open, using the "browse" function
16) Click open
17) When the file is password protected, you will be prompted for a password

2.3 SAVE WORKSPACE

18) To save a workspace, select File => Save workspace.
19) Select a directory, by using the "Browse" function, where the workspace file needs to be stored.
20) A couple of options is available:
    - Compress: Compresses the file to lower the required disk space
    - Password protection: Protect the file with a password, this makes the file unreadable when opening with a text editor.
21) Click "Save" and your workspace will be saved



3 MENU: DEFINE INTENDED ARCHITECTURE

3.0 MODULE TYPES AND RULE TYPES[12]

HUSACCT stands out in its support of semantically rich modular architectures, which are very common in practice. A semantically rich modular architecture (SRMA) includes modules of semantically different types, while a variety of types of rules may constrain the modules. As an example of an SRMA, Figure 1 shows a small part of an architecture model of one of the systems at an airport. This system is used to manage the state and services of human interaction points where customers communicate with baggage handling machines, self-service check-in units, et cetera. Examples in the rest of this document refer to elements in Figure 1.

Figure 1 shows UML icons for three semantically different types of modules: packages, components and interfaces. Layers are the fourth module type in the model (indicated by lines, since layers are not supported by UML). Finally, Spring and Hibernate represent the fifth type of module in the model: external system.

UML dependency relations in this example indicate is-only-allowed-to use rules; for instance, module HiWebApp is only allowed to use the modules HiForms and HimInterface, no others. Some other rules are not visible in the diagram. For example, rules related to the layered style, like “Technology Layer is not allowed to use Interaction Layer. Other examples of not visible rules are naming rules and rules inherent to components with interfaces.

Common Module and Rule Types

To enable compliance checks of SRMAs, rich sets of module and rule types should be supported. In a previous study [10], we presented a classification of common module types and common rule types. In this study, we use these common types as functional requirements to SRMA support. The next sub-sections describe these common module and rule types concisely to enhance practical understanding before the metamodel is presented. For a more in-depth discussion of the common module and rule types, we refer to our previous study.

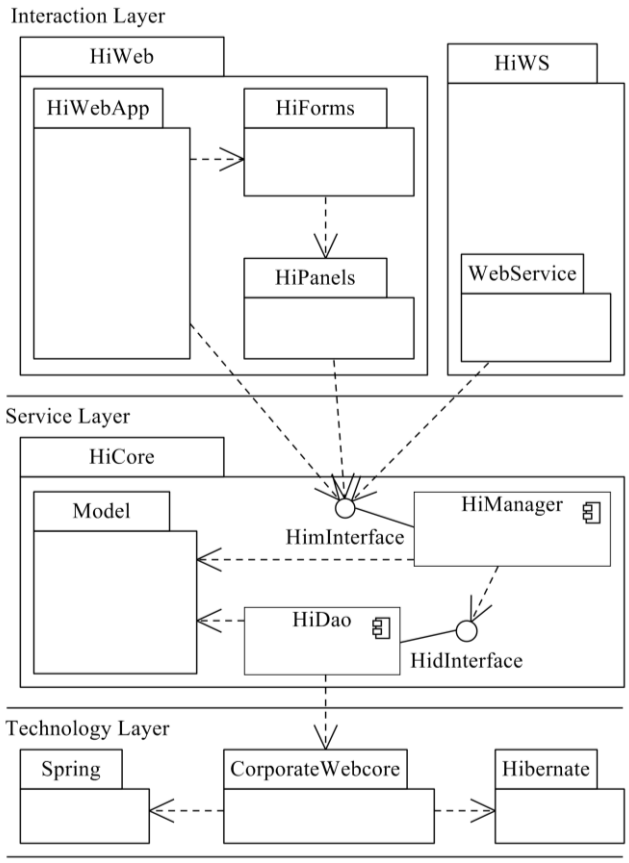


Figure 1. Example of an SRMA model

3.0.1 COMMON MODULE TYPES

SRMAs may contain modules of different types, with very different semantics. HUSACCT provides support for the following common types (of modules relevant for static ACC):

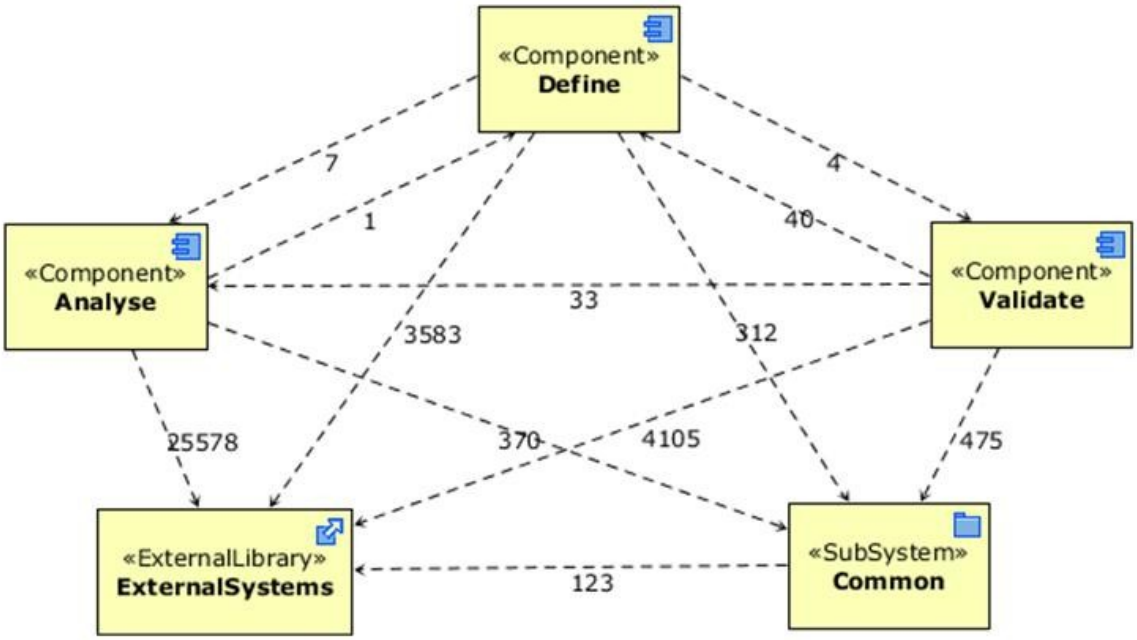
Subsystems represent units in the system design with clearly assigned responsibilities, but with no additional semantics. Comparable terms are logical cluster, or packages.

Layers represent units in the system design with additional semantics. Layers have a hierarchical level and constraints on the relations between the layers. We cite Larman [5], who summarizes the essence of a layered design as “the large-scale logical structure of a system, organized into discrete layers of distinct, related responsibilities. Collaboration and coupling is from higher to lower layers.”

Components within software architecture are designed as autonomous units within a system. The term component is defined in different ways in the field of software engineering. In our use, a component within a modular architecture covers a specific knowledge area, provides its services via an interface and hides its internals (in line with the system decomposition criteria of Parnas [6]). Consequently, a component differs from a logical cluster in the fact that it has a Interface sub module and hides its internals. Since our definition of component is intended for modular architectures, it does not include runtime behavior as in the “component and connector view” of architecture [2].

Interfaces are related to a component and act as facades, as described by the facade pattern [3]. An Interface at design level differs from the Java interface. An Interface may be mapped to multiple elements at implementation level, like Java interface classes, exception classes and data transfer classes.

External Library represents platform and infrastructural libraries or components used by the target system. HUSACCT’s ACC support includes the identification of usage of external systems and checks on constraints regarding their usage.



3.0.2 COMMON RULE TYPES

Modular architectures may contain rules of different types, where each rule type characterizes another kind of constraint on a module. These constraints are categorized in literature [2], [8] as properties and relationships. Our inventory of architectural rule types, in principle verifiable by static ACC, resulted in two categories related to properties and relationships: Property rule types and Relation rule types. The rule types supported by HUSACCT are described and exemplified in Table 1.

Property rule types constrain a certain characteristic of the elements included in the module and their sub modules. Clements et al. [2] distinguish the following properties per module: Name, Responsibility, Visibility, and Implementation information. We identified rule types associated to these properties and named them accordingly, except two types (Facade convention, Inheritance convention), which represent the property Implementation information.

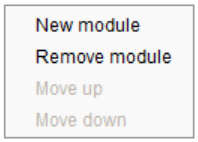
Relation rule types specify whether a module A is allowed to use a module B. The basic types of rules are "is allowed to use" and "is not allowed to use". However, we encountered useful specializations of both basic types, which we included in the classification. Table 1 shows the two included specializations of "Is not allowed to use" (both specific for layers), and the three specializations of "is allowed to use".

Table 1. Common rule types

Category\Type of Rule	Description (D), Example (E)	Ref [1]
Property rule types		
Naming convention	D: The names of the elements of the module must adhere to the specified standard. E: HiDao elements must have suffix DAO in their name.	[7]
Visibility convention	D: All elements of the module have the specified or a more restricting visibility. E: HiManager classes have package visibility or lower.	[2]
Facade convention	D: No incoming usage of the module is allowed, except via the facade. E: HiManager may be accessed only via HimInterface.	[3]
Inheritance convention	D: All elements of the module are sub classess of the specified super class. E: HiDao classes must extend CorporateWebCore.Dao.GenEntityDao.	[7]
Relation rule types		
Is not allowed to use	D: No element of the module is allowed to use the specified to-module. E: HiPanels is not allowed to use HiWS.	[4]
Back call ban (specific for layers)	D: No element of the layer is allowed to use a higher-level layer. E: Service Layer is not allowed to use the Interaction Layer.	[14]
Skip call ban (specific for layers)	D: No element of the layer is allowed to use a lower layer that is more than one level lower. E: Interaction Layer is not allowed to use the Infrastructure Layer.	[14]
Is allowed to use	D: All elements of the module are allowed to use the specified to-module. E: HiWebApp is allowed to use HiForms.	[2]
Is only allowed to use	D: No element of the module is allowed to use other than the specified to-module(s). E: HiForms is only allowed to use HiPanels.	[7]
Is the only module allowed to use	D: No elements, outside the selected module(s) are allowed to use the specified to-module. E: HiDao is the only module allowed to use CorporateWebcore.	[7]
Must use	D: At least one element of the module must use the specified to-module. E: HiDao must use CorporateWebcore.	[4]

[1] Ref= primary literature reference

3.1 DEFINE INTENDED ARCHITECTURE



This section will elaborate on the common tasks that you will need to perform to define an intended architecture. Keep in mind that most of these functionalities can also be accessed by using the right-click mouse button.

3.1.1 OVERVIEW

The define view is split into 5 areas, see the figure below. Each area is described below.

#1 Module Hierarchy

This area provides a decomposition view on the intended architecture. The hierarchy of modules is shown, while modules may be added, selected, and edited. You can select a module by clicking on it.

#2 Module Properties

This area shows the properties of the selected module. You can change the module's name, description, and type.

#3 Assigned Software Units

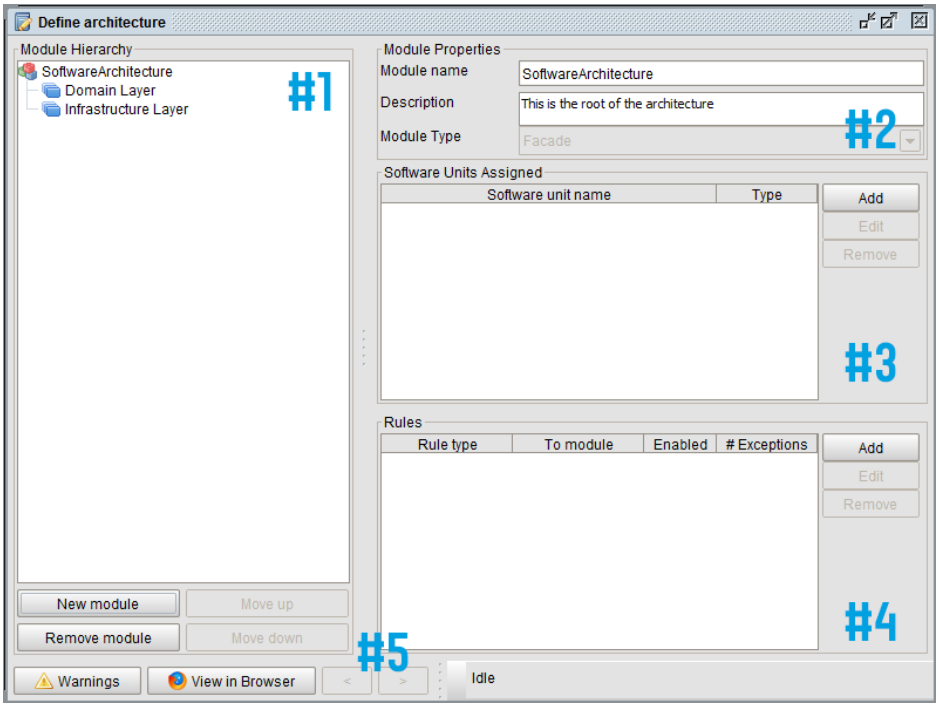
This area shows the software units in the implemented architecture, which are assigned to the selected module. The buttons are only enabled if a module is selected.

#4 Rules

This area shows the rules, which are defined for the selected module. The buttons are only enabled if a module is selected.

#5 Toolbar

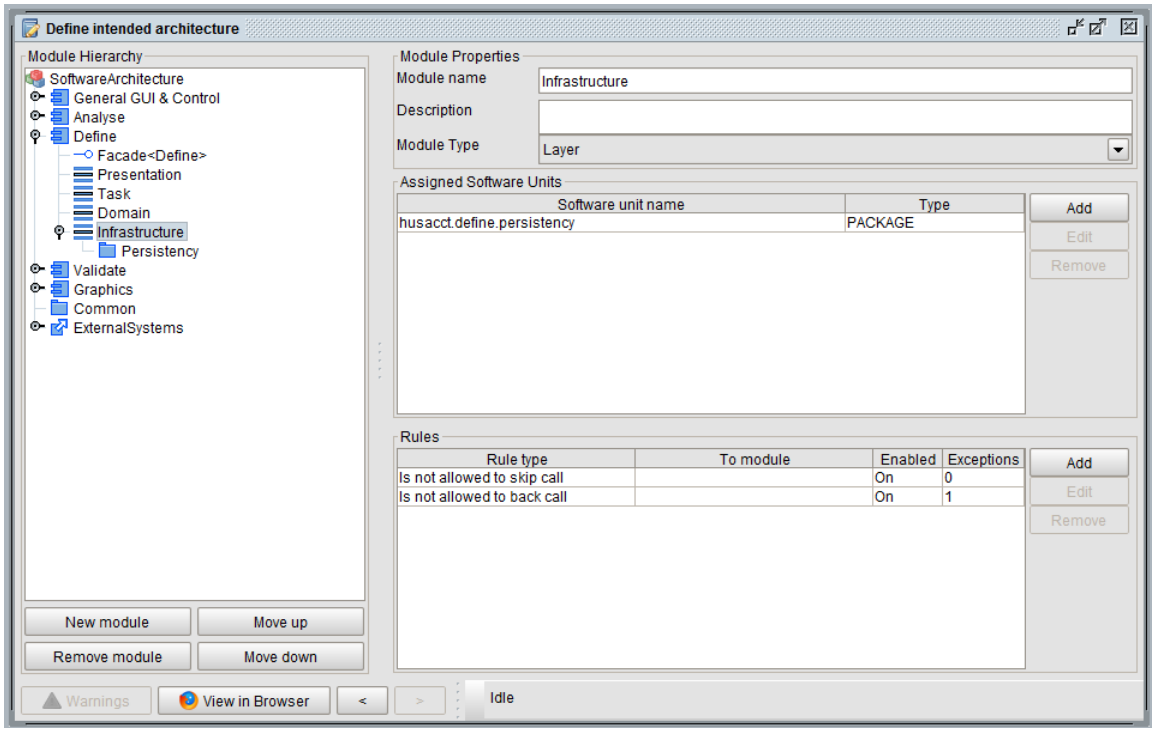
The toolbar has some features that are not strictly necessary to define the architecture, but which provide additional support. (Currently, most of the functionality below is disabled.) The first button is the warnings button. When activated, a dialog appears with warnings about inconsistencies in your intended architecture. The second button, 'View in Browser', will activate a HTML-report/overview with all modules, applied rules and assigned software units. Next, there are undo and redo buttons. Made a mistake? Undo it!



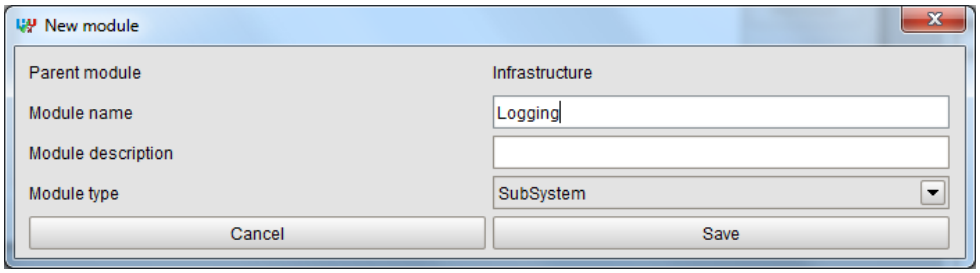
3.1.2 ADD MODULES

Accomplish the following steps to add a module to the logical architecture:

1. Select nothing or select the root node if you want to create a module to the root of the architecture. Otherwise select the module you wish to create a child module for.

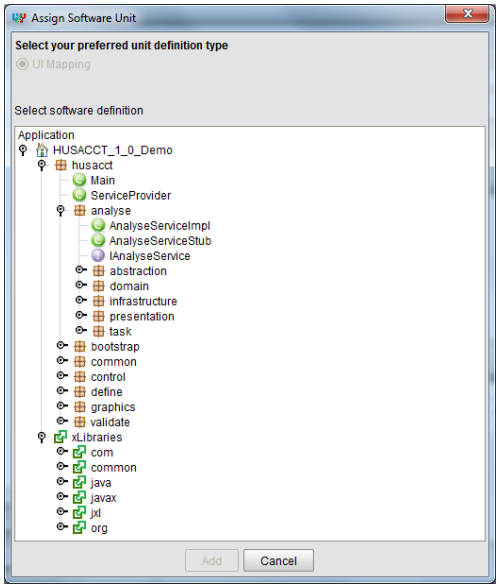


2. Click on the “New Module” button.



3. Enter a module name. Optional: Enter a description.
4. Select a module type.
5. Click on the “Save” button and the new module will be added to the module hierarchy.

3.1.3 ASSIGN SOFTWARE UNITS



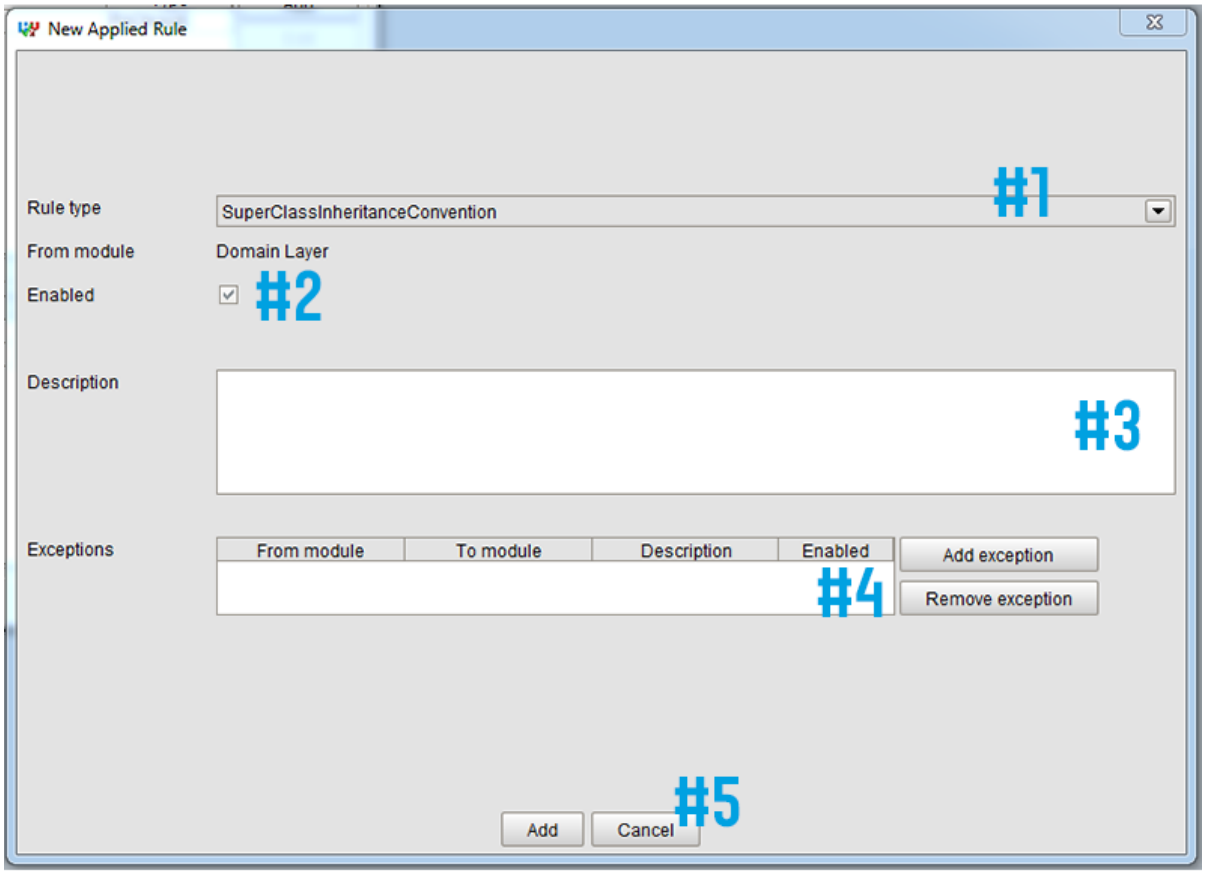
Note: To be able to assign software units to a module, the application needs to be analysed in advance.
Accomplish the following steps to assign one or several software units to a module:

1. Select the module you wish to map.
2. Click on the “Add” button in the *Software Units Assigned* section.
3. Select the software unit you wish to map to the selected module. You can use the “Shift” key or “Ctrl” key to select multiple software units.
4. Click on the “Add” button to assign the selected software units to the selected module.

3.1.4 ADD RULES

Accomplish the following steps to add a rule to a module:

1. Select the module you wish to create a rule for.
2. Click on the “Add” button in the *Rules* section and the New Applied Rule form appears.



3. Select the rule type you wish to create (*#1 in the figure*).
4. In case of relation rules: Select the To-module of which the usage is constrained.
5. By default the rule is Enabled, but the rule may be disabled (#2 in the figure); temporarily, or continuously, e.g. in case of generated default rules.
6. Optional: Enter a description for this rule (#3 in the figure).
7. Optional: Add exception rules (#4 in the figure). Read subsection "Add Exceptions to a Rule".
8. Enter an Expression, if required. Read subsection "Set Filter and/or Expression to a Rule".
   Rules of the following rule types require an Expression: Naming convention, Visibility convention.
9. Click on the "Add" button to add this rule with all its exception rules (#5 in the figure).

3.1.5 ADD EXCEPTIONS TO A RULE

Accomplish the following steps to add an exception to a rule:

1. Open the rule details panel. You can do this by adding a new rule, or by selecting an existing rule and pressing the "Edit" button in the *Rules* panel.
2. Press the "Add exception" button.
3. Select the rule type of the exception rule you wish to create. However, most rules have only one exception rule type available: is allowed to use.
4. Select the From-module and/or To-module.
5. Enable or disable this exception rule if needed.
6. Optional: Enter a description for this rule.
7. Optional: Configure the Violation Types.
8. Fill in any details required by the rule type. For example, the naming convention rule requires you to enter a regex.
9. Click on the "Add" button to add this exception rule to the main rules.

3.1.6 SET FILTER AND/OR EXPRESSION TO A RULE

3.1.6.1 Naming convention

In case of a rule of this type, all the class names and/or package names within the selected module should meet a specified Expression. Expressions have to be combinations of: 1) a text (case-sensitive) that has to be part of the name; and 2) *, which represent unconstrained text.
Examples are provided in the table below.

Use 'Configure Filter' to specify whether the rule should be applied to packages and/or to classes. By default, none of these two are selected. So, adding a naming convention rule without setting these filter options may result in false negatives (non-reported violations).
Note: If the panel popping up after activating 'Configure Filter' does not show selection options:
1) Save the rule; 2) Select and Edit the rule, and activate 'Configure Filter' again.

EXPRESSION	VALIDATES	RESULT
Friends	domain.locationbased.foursquare.History	False
	domain.locationbased.latitude.Friends	True
	infrastructure.socialmedia.locationbased.foursquare.FriendsDAO	True
	infrastructure.socialmedia.locationbased.foursquare.MyFriendsDAO	True

***Account**	domain.locationbased.foursquare.MyAccount	True
	domain.locationbased.latitude.Map	False
	infrastructure.socialmedia.locationbased.foursquare.AccountDAO	False
DAO *	infrastructure.socialmedia.locationbased.foursquare.DAOFourSquare	True
	infrastructure.socialmedia.locationbased.foursquare.IMap	False
	domain.locationbased.foursquare.History	False

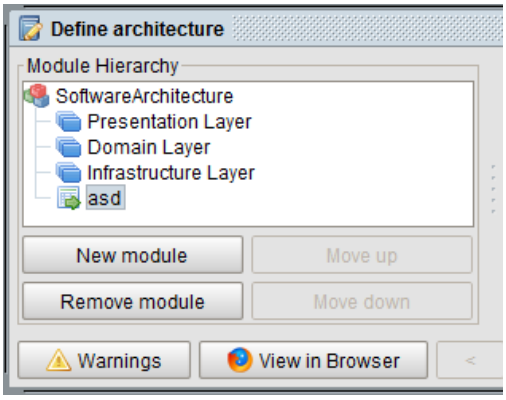
The first Expression in the table enforces that a class/packages must contain the word 'Friends'.
The second Expression enforces that a class/package must end with the word 'Account').
The third Expression enforces that all the classes and/or packages must start with the word 'DAO'.

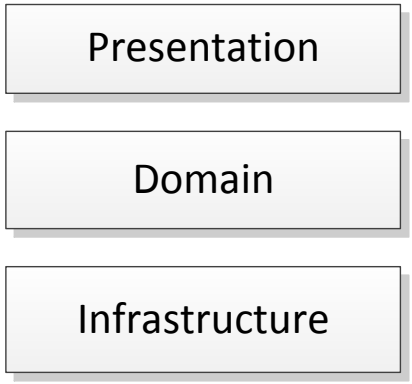
3.1.6.2 Visibility convention

In case of a rule of this type, the visibility of all the classes and packages in the specified module must have the specified level of visibility, or lower. Use 'Configure Filter' to specify the level of visibility allowed as maximum. By default, no visibility is selected. So, adding a visibility convention rule without setting these filter options may result in false negatives (non-reported violations).
Note: If the panel popping up after activating 'Configure Filter' does not show selection options:
1) Save the rule; 2) Select and Edit the rule, and activate 'Configure Filter' again.

3.1.7 MOVE LAYERS

For modules of type "Layer" the hierarchical position is an important attribute. To other types of modules, the hierarchical position is of no importance.
The module hierarchy in the figures below is equivalent.



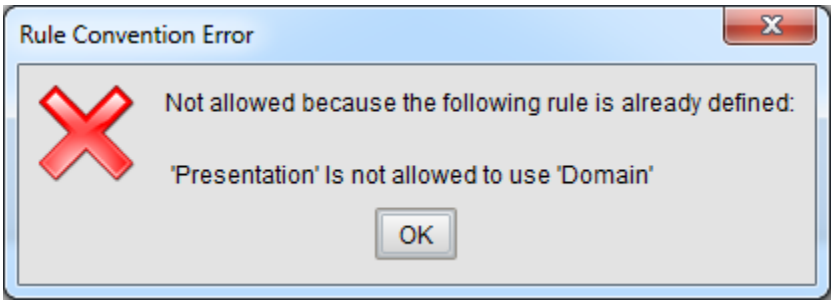


To move a layer up or down you will need to:

1. Select a layer.
2. Click on the "Move up" or "Move down" button in the *Module Hierarchy* section.

3.1.8 CONFLICTING RULES

It is not possible to define conflicting rules. For instance, if you were to create a rule that would state that the “Presentation” module is not allowed to use the “Domain” module. It would be impossible to define a rule that would state that the “Presentation” module must use the “Domain” module. If you would try to, this will result in the following error message.



Here is a list of all rules that cannot be defined if a rule of a certain type is already in place.

USE CASE	FORBIDDEN WHEN FOLLOWING RULE IS DEFINED
Naming convention	• “Naming convention” rule in the same module
Visibility convention	• “Visibility convention” rule in the same module
Subclass convention	• “Subclass convention”: rule in the same module • Same checks as a “must use” rule
Interface convention	• “Interface convention” rule in the same module • Same checks as a “must use” rule
Is not allowed to use	• “Is only allowed to use”, “is only module allowed to use”, “Is allowed to use” or “must use” rule from the selected module to the selected “module to”
Is only allowed to use	• “Is not allowed to use” rule from this module to the selected “module to” • “Is only allowed to use”, “is only module allowed to use”, “is allowed to use” or “must use” rule from this module to other then the selected “module to” • “Is only module allowed to use” rule from other then the selected module to the selected “module to”
Is only module allowed to use	• “Is not allowed to use” rule from this module to the selected “module to” • “Is only module allowed to use”, “is only module allowed to use”, “is allowed to use” or “must use” rule from other then the selected module to the selected “module to”
Is allowed to use	• “Is not allowed to use” rule from this module to the selected “module to” • “Is only allowed to use” rule from this module to other then the selected “module to” • “Is only module allowed to use” rule from other then the selected module to the selected “module to”
Must use	• “Is not allowed to use” rule from this module to the selected “module to” • “Is only allowed to use” rule from this module to other then the selected “module to”

	• "Is only module allowed to use" rule from other then the selected module to the selected "module to"
Skip call	• Same checks as a "is not allowed to use" rule for the 2nd layer below the selected layer, and each layer below this 2nd layer. You can see this layer as the selected "module to" layer.
Back call	• Same checks as a "is not allowed to use" rule for each layer above the selected layer. You can see this layer as the selected "module to" layer.

3.1.9 VIEW INTENDED ARCHITECTURE IN BROWSER

When you click on the 'View in Browser'-button on the main screen of the define component, a report will be generated. This report consists out of an overview of modules, applied rules and software units and of a table with all the modules with their software units and applied rules. For example, see the figures below.

An overview of your architecture

There is a total of:

- 6 Modules;
- 8 Applied Rules (8 active);
- 0 Software Units.

[+] Expand All

Module	Software Unit	Applied Rule
Presentation Layer		IsNotAllowedToMakeSkipCall IsNotAllowedToMakeBackCall
Domain Layer		IsNotAllowedToMakeSkipCall IsNotAllowedToMakeBackCall
Infrastructure Layer		IsNotAllowedToMakeSkipCall IsNotAllowedToMakeBackCall

An overview of your architecture

There is a total of:

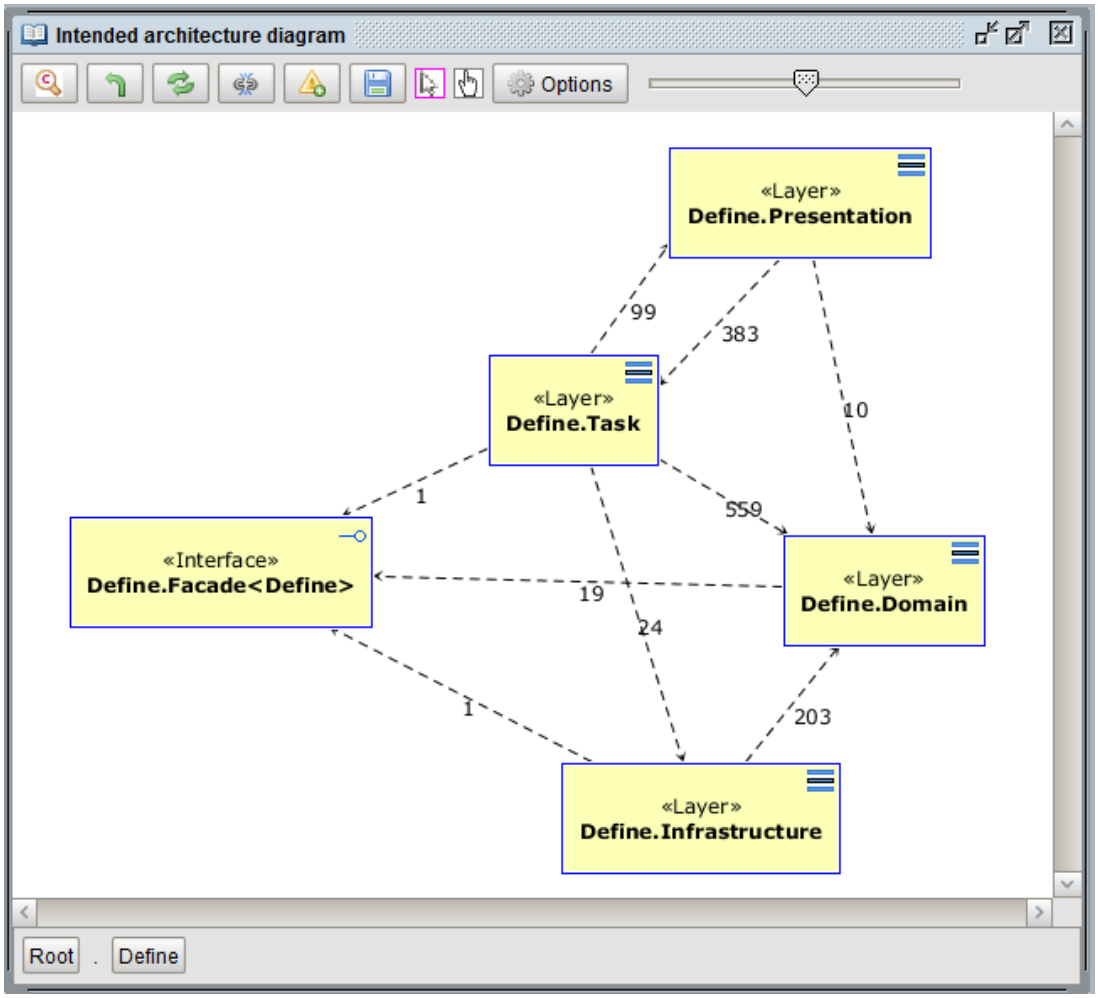
- 6 Modules;
- 8 Applied Rules (6 active);
- 0 Software Units.

[-] Collapse All

Module	Software Unit	Applied Rule
Presentation Layer		IsNotAllowedToMakeSkipCall IsNotAllowedToMakeBackCall
qwe		VisibilityConvention
Domain Layer		IsNotAllowedToMakeSkipCall IsNotAllowedToMakeBackCall
Infrastructure Layer		IsNotAllowedToMakeSkipCall IsNotAllowedToMakeBackCall
asdas		FacadeConvention
Facade		

3.2 INTENDED ARCHITECTURE DIAGRAM

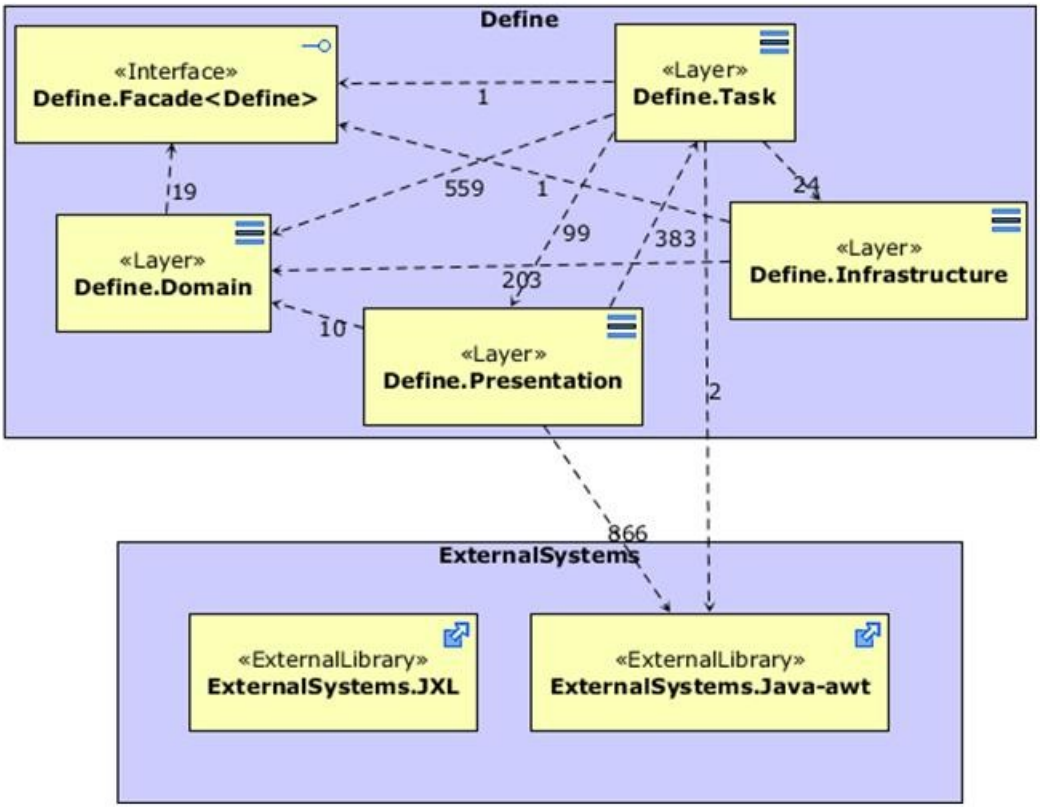
An intended architecture diagram shows the modules of the intended architecture at a certain hierarchical level. For each module the name and modules type are shown, and (as soon as implemented software units are assigned to the modules) dependency arrows, with the number of detected dependencies, are shown between the modules.



The following functionality may be used to adjust and export diagrams (for more instruction and details, visit the section on Implemented architecture diagram):

- Move modules within the diagram.
- Hide specific modules, and restore these modules, if needed.
- Zoom in on a selected module, or on several selected modules (multi zoom, see the example below). Furthermore: zoom out, refresh.
- Export a diagram to an image file (png file).
- Optional: show dependencies, show violations, show thickness of dependency arrow relative to the number of dependencies.
- Pan function.

Example of a multi zoom diagram:



3.3 IMPORT AND EXPORT ARCHITECTURE

The definition of the intended architecture may be exported as an xml file. It may be reused by an import of the file in different workspaces. The modules and rules are reusable, but the assignment of implemented software units to modules must likely be changed.

4 MENU: ANALYSE IMPLEMENTED ARCHITECTURE

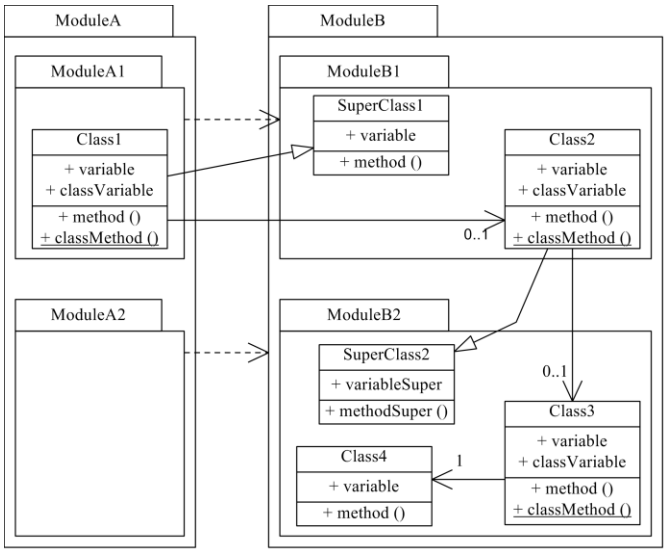
4.0 DEPENDENCY ANALYSIS [11]

Dependency analysis is "the process of determining a program's dependences"[9]. Various types of dependencies are distinguished in literature. Callo Arias et al. [1]consider that all types fit into three main categories: structural dependencies, behavioral dependencies, and traceability dependencies. The category of structural dependencies, dependencies among parts of a system, is of interest here, since static analysis tools focus on dependencies that can be found by inspecting the source code.

4.0.1 EXAMPLE OF A MODULAR ARCHITECTURE

The different types of dependency reported by HUSACCT are specified in the next subsections. These dependency types are illustrated on the basis of a modular architecture in UML notation, shown in the figure below. In this diagram, two modules, ModuleA and ModuleB, are shown, each with two submodules. The classes in the submodules are related via associations, showing for instance that an instance of Class1 may know several instances of Class 2. The dependency arrows show that ModuleA is allowed to use ModuleB1 and that Module A2 is allowed to use ModuleB. However, not all rules are visible. The following list shows the full set of relationship rules:

- ModuleA1 is allowed to use ModuleB1;
- ModuleA2 is allowed to use ModuleB, so also both sub modules, ModuleB1 and ModuleB2;
- ModuleA1 is not allowed to use ModuleB2;
- The submodules of ModuleA are allowed to use each other. The same type of rule applies to ModuleB.



Example of a modular architecture in UML notation.

4.0.2 DIRECT STRUCTURAL DEPENDENCY TYPES

A dependency between two modules is *direct*, if there is an explicit reference to the to-class in the from-class (or in a super class of the from-class). For example, ModuleA in the figure depends on ModuleB, because a class in ModuleA1 uses a class in ModuleB1 with an explicit reference to that class. In Java, a preceding specification of an import command is required.
An overview of the direct structural dependency types is shown in the first table below, together with an example per sub category.

4.0.3 INDIRECT STRUCTURAL DEPENDENCY

A dependency relation is indirect, when the dependency exists transitively through an intermediate module. For example, ModuleA1 in the figure depends on ModuleB2 via ModuleB1. In that case, a class uses another class without an explicit reference to that class, so in Java no import command is required.
An overview of the identified indirect structural dependency types is shown in the second table below, together with an example per sub category.

DIRECT STRUCTURAL DEPENDENCY TYPES IN THE TEST

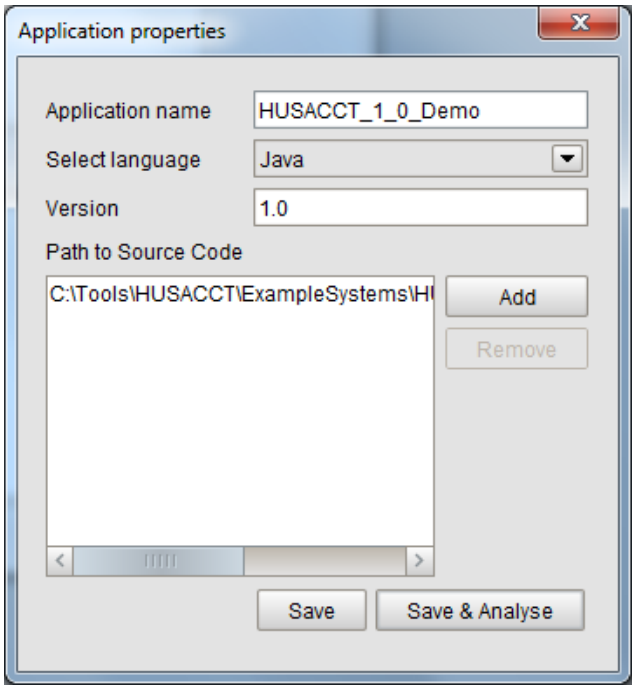
Sub Category/Dep. Type	**Example Code** (from Class1 in the fig.)
Import Class import	Import ModuleB.ModuleB1.Class2;
Declaration Instance, Class variable; Parameter; Return type.	private Class2 linkToC2;
Call Instance, Class method; Constructor.	public String variable; variable = linkToC2.method();
Access Instance, Class variable; Object reference.	variable = linkToC2.variable;
Inheritance Extends class, Implements interface	public class Class1 extends SuperClass1 { }
Annotation Class annotation	@Class2

INDIRECT STRUCTURAL DEPENDENCY TYPES IN THE TEST

Sub Category/Dep. Type	**Example Code** (from Class1 in the fig.)
Call Instance method; Class method.	public String variable; variable = linkToC2.linkToC3.method();
Access Instance, Class variable; Object reference.	variable = linkToC2.linkToC3.variable;
Inheritance Extends – extends; Access inherited variable;	variable = linkToC2.variableSuper();

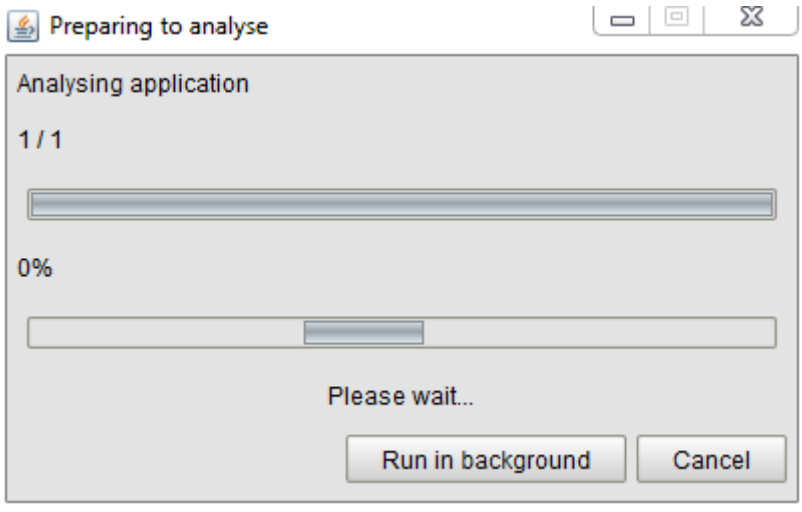
4.1 APPLICATION PROPERTIES

This option may be used to edit the application properties, like name, version, and path.



4.2 ANALYSE APPLICATION

When you run HUSACCT and create a new workspace you have the option to directly analyse a project. There is no difference between direct analysing and analysing on a later moment, when analysing is started the previous analysed information is cleared and the code will be completely reanalysed. Analysing a project can take up some time. Obviously, when the project contains many files it might take tens of seconds, while a small project is analysed within a second.



In the screenshot above you notice two loading bars which display the progress of the analysing. The top bar is implemented for future extensibility for analysing multiple projects. The lower bar is for the

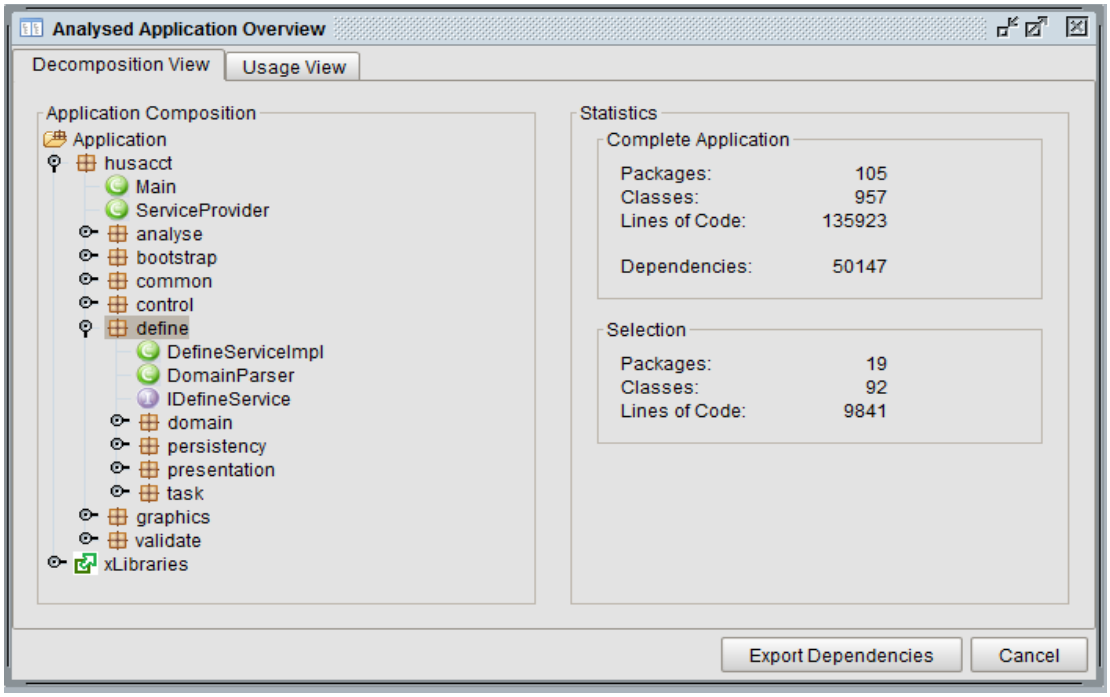
progress of the analysis of the current project. This bar starts running after the initial analysis process has finished and the repository is filled with raw data. Thereafter, hierarchical structures, external libraries and dependencies are derived from the raw data, and a dependency cache is build up.

4.3 ANALYSED APPLICATION OVERVIEW

The “Analysed Application Overview” helps you to study the decomposition of the software units of the implemented application, and the dependencies between these units. Two views are provided, which provide insight into the implemented architecture; useful for architecture reconstruction work.

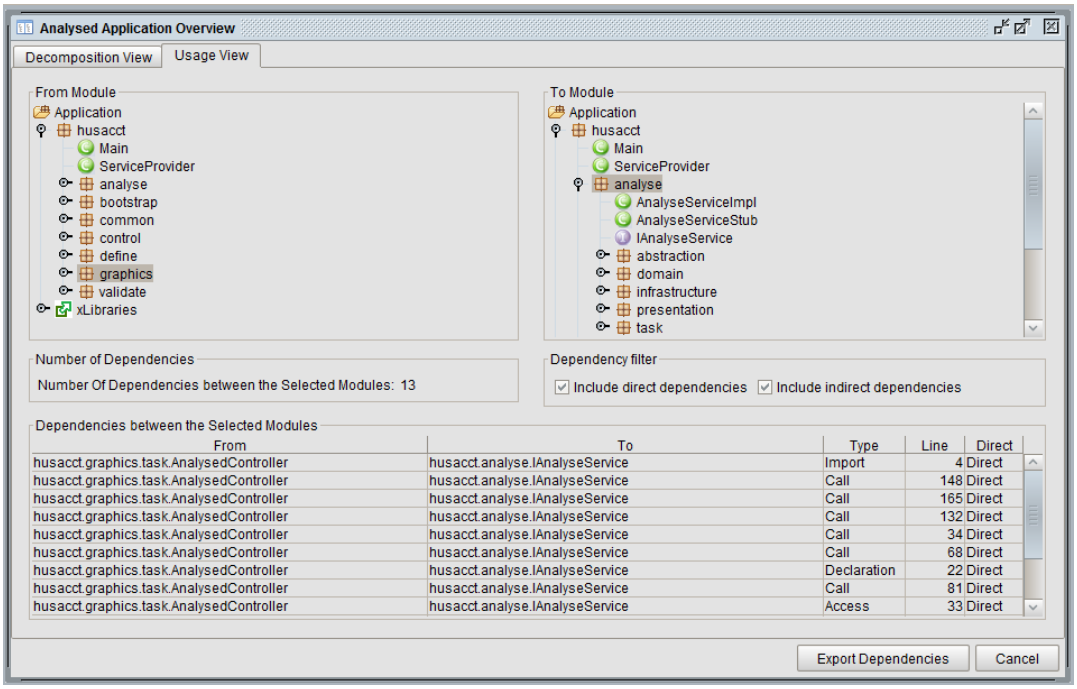
4.3.1 DECOMPOSITION VIEW

The “Decomposition View” is designed to let you study the hierarchical structure of the application and the used external systems. Software units may be selected and opened. Statistical information is shown for the application as a whole. Furthermore, when a unit is selected, statistical information about this unit is shown.
Packages: Number of packages, e.g. within a selected software unit or one of its child units.
Classes: Number of classes, e.g. within a selected software unit or one of its child units.
Lines of codes: Total number of lines of code (including comment and blank lines) within the classes, e.g. within a selected software unit or one of its child units.



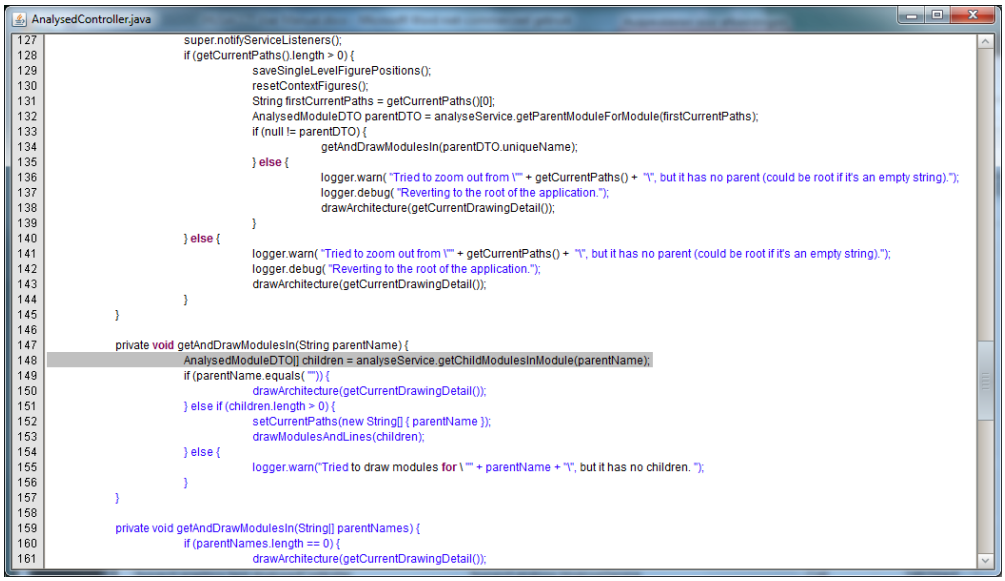
4.3.2 USAGE VIEW

The “Usage View” is designed to study the dependencies between the software units. Select modules on the left (the From Module) and right (the To Module). If there are dependencies between these modules, the total number of dependencies is shown, while the bottom part of the form will show with detailed information about each dependency.
The Dependency Filter may be used to filter on direct or indirect dependencies.



4.3.3 CODE VIEWER

The code viewer allows direct inspection of the source code. Activate the code viewer by a double click on the dependency in the dependency table. The line which includes the dependency is highlighted.

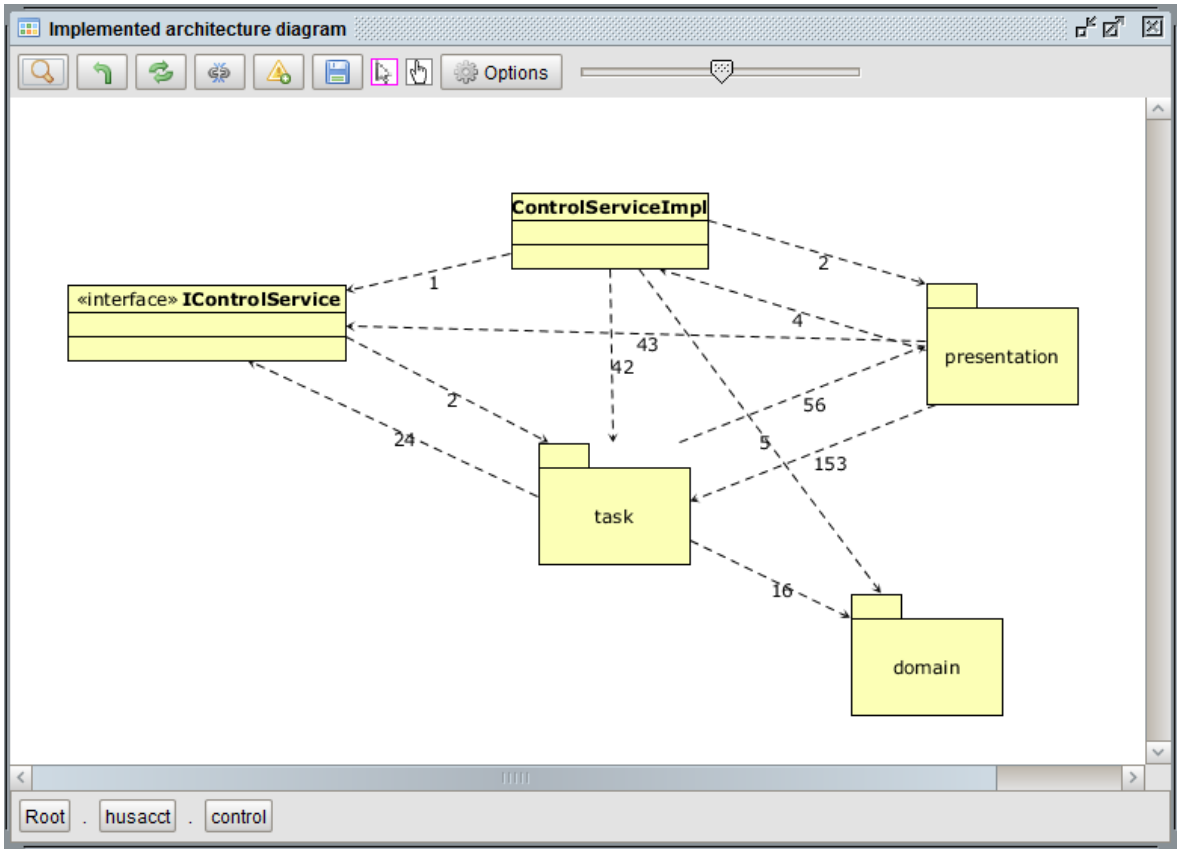


4.4 IMPLEMENTED ARCHITECTURE DIAGRAM

An implemented architecture diagram shows the implemented software units (packages, classes, interfaces) with their dependencies at a certain hierarchical level.
As visible in the picture below, of each software unit the name and type (in text or as icon) are shown, while dependency arrows with the number of detected dependencies are drawn from the unit to the units where it depends upon.

Modules may be selected, one or several concurrently, and moved to another position within the drawing canvas. That way, a comprehensible diagram may be created.
Note: The layout of the diagram is not stored and will be lost after zoom in, zoom out, refresh, or after a status change caused by actions under other menu options. So, if you want to save the layout, export the diagram as an image.

At the bottom of the editor, the path within the decomposition hierarchy is shown. As indicated, the diagram represents the contents of package husacct.control.



4.4.1 MENU BAR

At the top of the diagram editor, icons are shown which represent functionality to adjust and export a diagram. This functionality is explained below.

#2 Zoom Options
Left click to zoom, right click to select zoom options.

#3 Return
Return to the previous level of abstraction in the decomposition hierarchy.

#4 Refresh
Refresh the diagram. Changes to the layout will get lost.

#5 Dependencies
Click to toggle between including and not including dependency arrows.

#6 Violations
Click to toggle between including and not including violating dependency arrows.

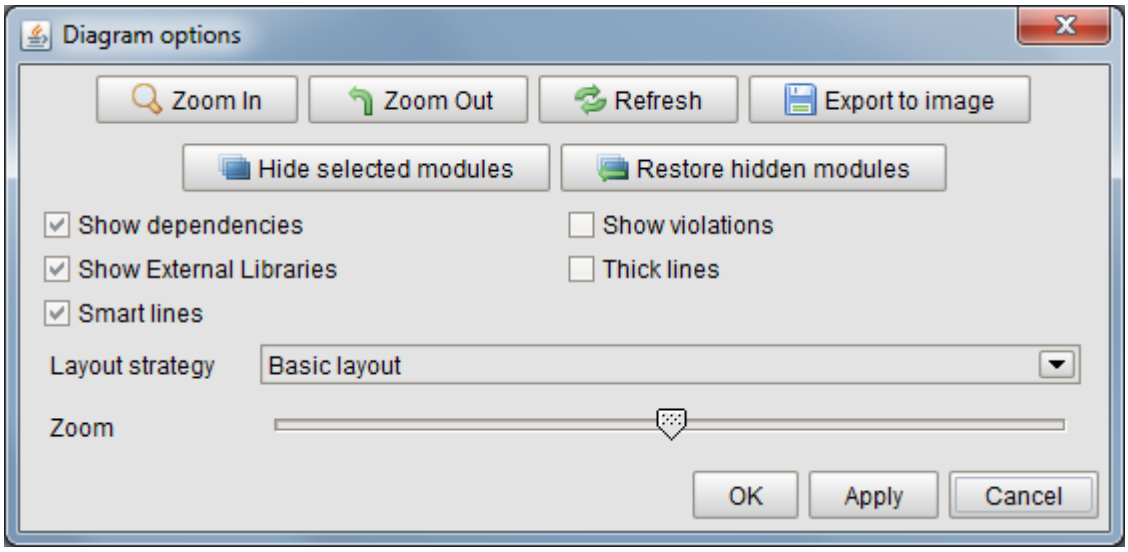
#7 Export diagram to file
Export the diagram to an image file.

#8 Mouse tools
Click the leftmost option for the select tool, the rightmost option for the pan tool.
The item bordered in cyan is the currently selected tool.
The pan option is useful if the diagram is larger than the shown part in the editor. In that case, select the pan option, click on the canvas, hold the left mouse button and move the mouse. As a result, the whole diagram will move with your mouse movement.

#9 Options
Click to open the options menu.

#10 Zoom slider
Slide to zoom in or zoom out.

4.4.2 OPTIONS DIALOG



The options dialog contains several functionalities also available in the menu bar.
The additional options are described below.

Hide selected modules
Removes the module or software unit temporarily from the diagram. It is not removed in the repository.

Restore hidden modules
The modules or software units, which were previously hidden, will be included again in the diagram.

Show External Libraries
This checkbox reflects whether or not external libraries will be displayed.
Select this option at root level.

Thick lines
When selected, the width of the dependency arrows will be shown relative to the number of dependencies represented by the arrow.

Smart lines
This option takes care of a smart distribution of the lines over the diagram.

Layout strategy
Select a suitable layout strategy. Currently only a few strategies are provided.

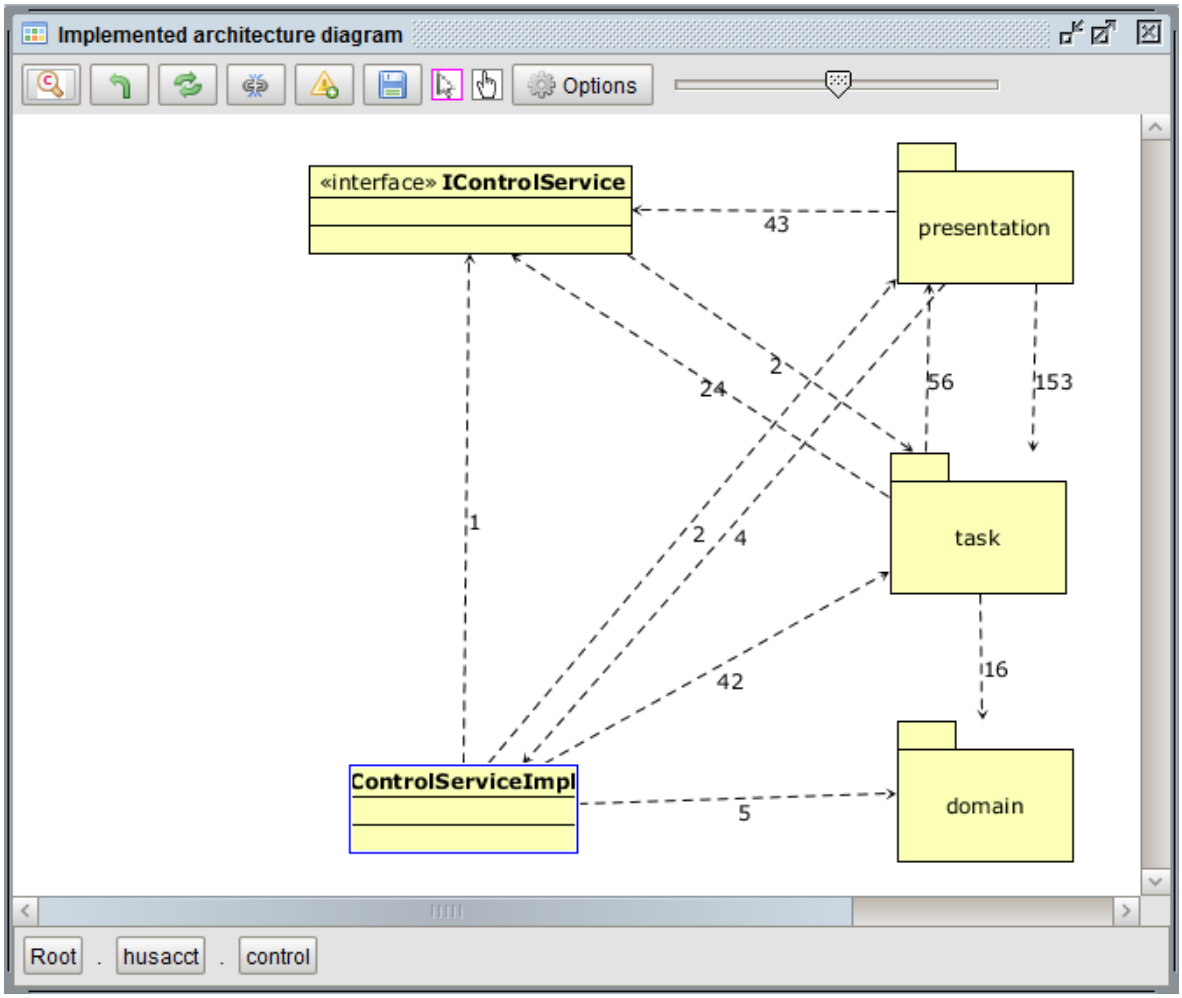
4.4.3 ZOOM OPTIONS

Several options to zoom in are available. These options are described below.

4.4.3.1 Default Zoom in

Default zoom in will show the contents of the selected module only in the decomposition diagram.
Procedure:
1) Check the zoom option icon in the menu bar. It should display a normal magnifying glass. If not, edit the zoom option setting with a right mouse click to 'Zoom In'.
2) Select a module in the diagram.
3) Zoom in: right mouse click => Zoom In, or click on menu bar icon, or click on icon in diagram options dialog.

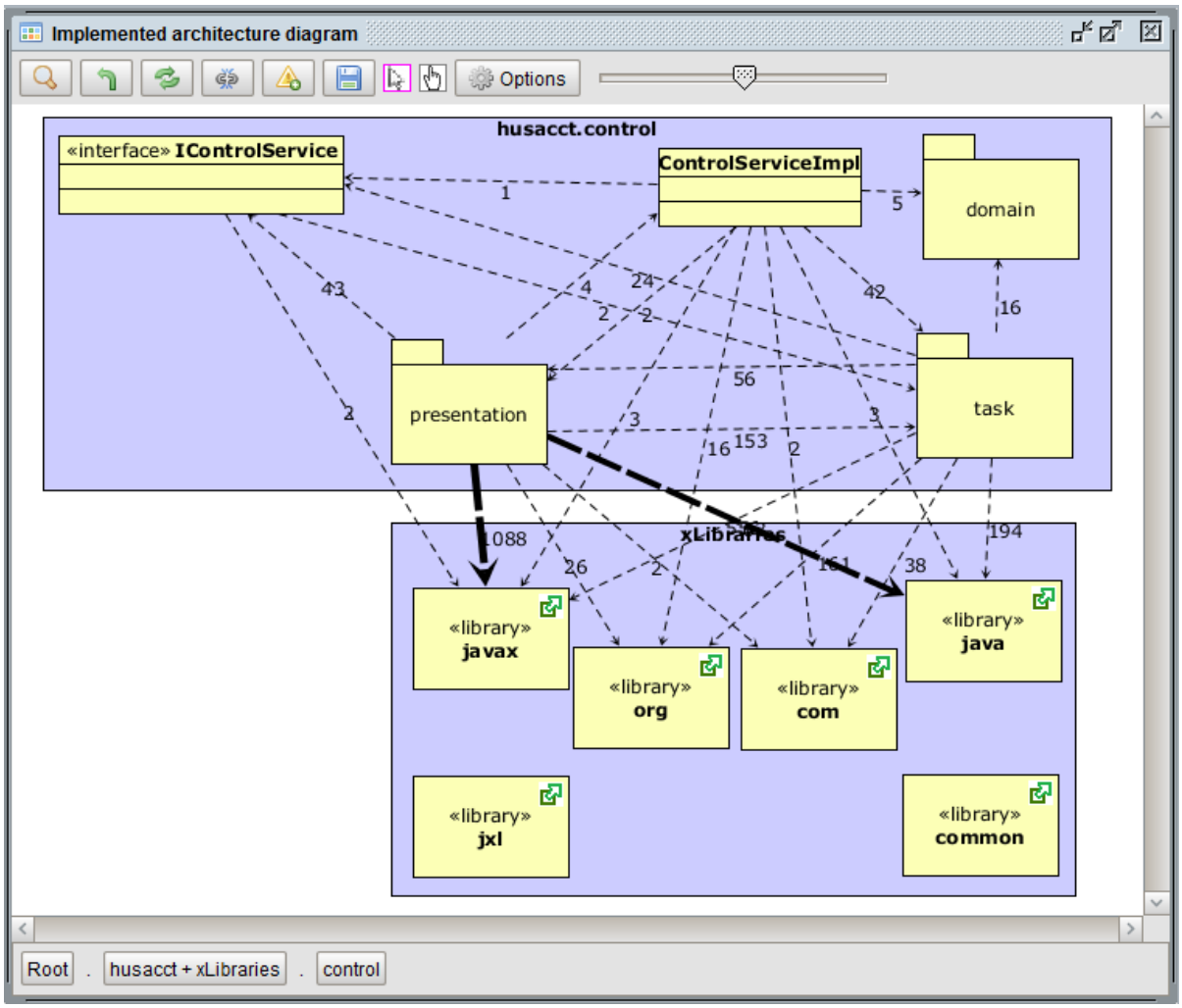
Example of a default zoom in on husacct.control.



4.4.3.2 Multi zoom in

Multi zoom in will show the contents of the selected modules in the decomposition diagram and the dependencies between the submodules of the selected modules.
The procedure is the same as with default zoom, with as difference that in step 2 two or three modules or software units may be selected (hold the shift key to select t.

Example of a multi zoom in on husacct.control, while xLibraries was selected as well.
Furthermore, the option 'Thick lines' is selected, and one software unit is hidden.

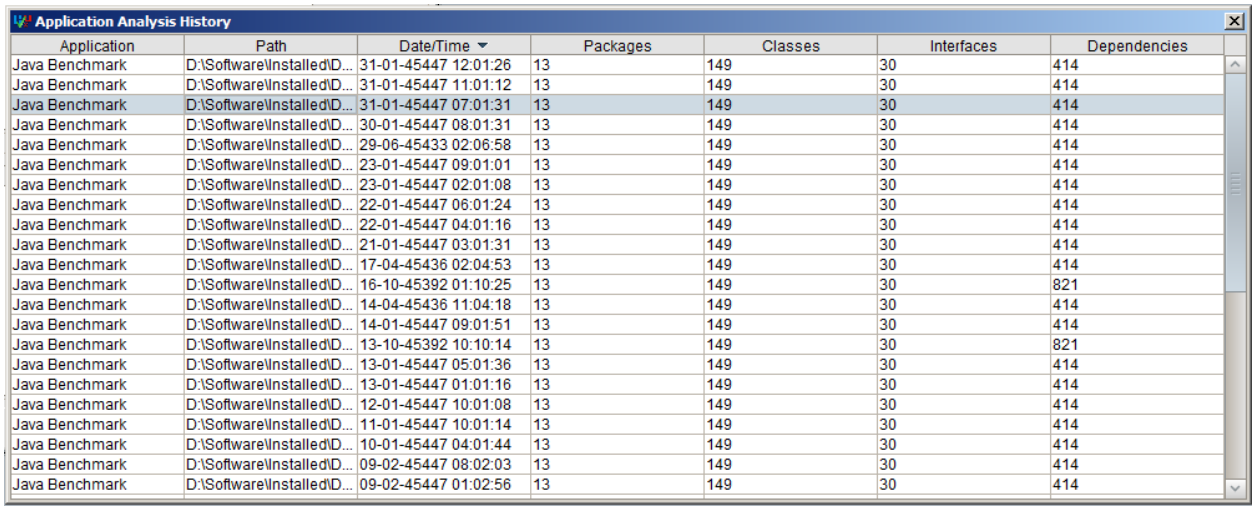


4.4.4 BROWSE DEPENDENCIES & VIEW CODE

Selecting a dependency arrow will activate a dependency (or violation) table, which shows the dependencies or violations represented by the arrow.
A double click on a dependency will activate the code viewer. Within the code, the (violating) line is highlighted.

4.5 ANALYSIS HISTORY

After analyzing any source code, HUSACCT saves information about the analysis in its App Data folder. This allows you to view the evolution of the software while your optimizing your code to get as few violations as possible. This analysis information looks like this:



Application	Path	Date/Time	Packages	Classes	Interfaces	Dependencies
Java Benchmark	D:\Software\Installed\D...	31-01-45447 12:01:26	13	149	30	414
Java Benchmark	D:\Software\Installed\D...	31-01-45447 11:01:12	13	149	30	414
Java Benchmark	D:\Software\Installed\D...	31-01-45447 07:01:31	13	149	30	414
Java Benchmark	D:\Software\Installed\D...	30-01-45447 08:01:31	13	149	30	414
Java Benchmark	D:\Software\Installed\D...	29-06-45433 02:06:58	13	149	30	414
Java Benchmark	D:\Software\Installed\D...	23-01-45447 09:01:01	13	149	30	414
Java Benchmark	D:\Software\Installed\D...	23-01-45447 02:01:08	13	149	30	414
Java Benchmark	D:\Software\Installed\D...	22-01-45447 06:01:24	13	149	30	414
Java Benchmark	D:\Software\Installed\D...	22-01-45447 04:01:16	13	149	30	414
Java Benchmark	D:\Software\Installed\D...	21-01-45447 03:01:31	13	149	30	414
Java Benchmark	D:\Software\Installed\D...	17-04-45436 02:04:53	13	149	30	414
Java Benchmark	D:\Software\Installed\D...	16-10-45392 01:10:25	13	149	30	821
Java Benchmark	D:\Software\Installed\D...	14-04-45436 11:04:18	13	149	30	414
Java Benchmark	D:\Software\Installed\D...	14-01-45447 09:01:51	13	149	30	414
Java Benchmark	D:\Software\Installed\D...	13-10-45392 10:10:14	13	149	30	821
Java Benchmark	D:\Software\Installed\D...	13-01-45447 05:01:36	13	149	30	414
Java Benchmark	D:\Software\Installed\D...	13-01-45447 01:01:16	13	149	30	414
Java Benchmark	D:\Software\Installed\D...	12-01-45447 10:01:08	13	149	30	414
Java Benchmark	D:\Software\Installed\D...	11-01-45447 10:01:14	13	149	30	414
Java Benchmark	D:\Software\Installed\D...	10-01-45447 04:01:44	13	149	30	414
Java Benchmark	D:\Software\Installed\D...	09-02-45447 08:02:03	13	149	30	414
Java Benchmark	D:\Software\Installed\D...	09-02-45447 01:02:56	13	149	30	414

4.6 EXPORT DEPENDENCIES

Select this menu option to create a spreadsheet file with all the detected dependencies within the application. Specify the location where the file will be stored, enter a file name and click on ‘Export’.

5 MENU: VALIDATE CONFORMANCE

5.1 VALIDATE NOW

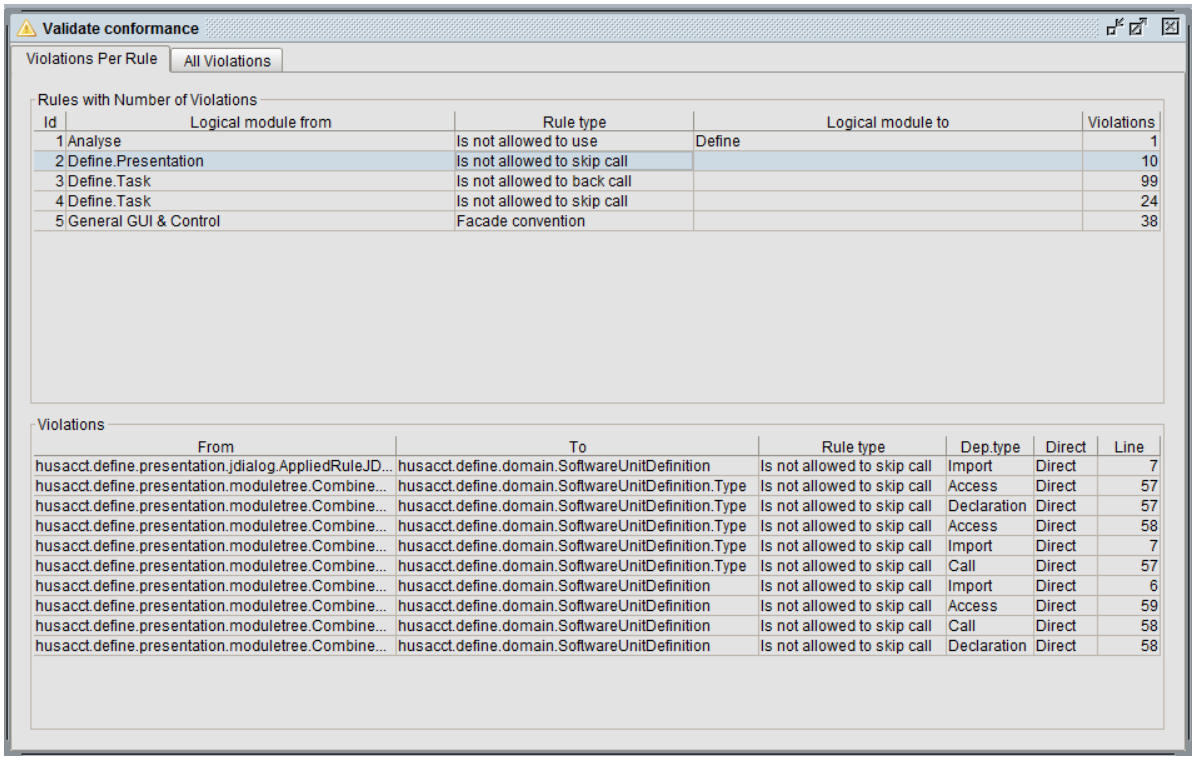
Validate conformance checks if rules defined in the intended architecture are violated in the implemented architecture.
The following preconditions apply:

- The source code is analyzed;
- Logical modules are defined;
- The classes or packages of the source code are assigned to the defined logical modules;
- The intended architecture contains rules.

When the validation process has finished, the results are shown in the from 'Validate conformance', with two tabs: 1) Violations per Rule; and 2) All Violations.

5.1.1 VIOLATIONS PER RULE

This view provides a summary of the violations against the rules.
The upper table shows the violated rules and the number of violations per rule. When a rule in this table is selected, the violations are shown in the bottom table.
A double click on a dependency will activate the code viewer. Within the code, the violating line is highlighted in red.



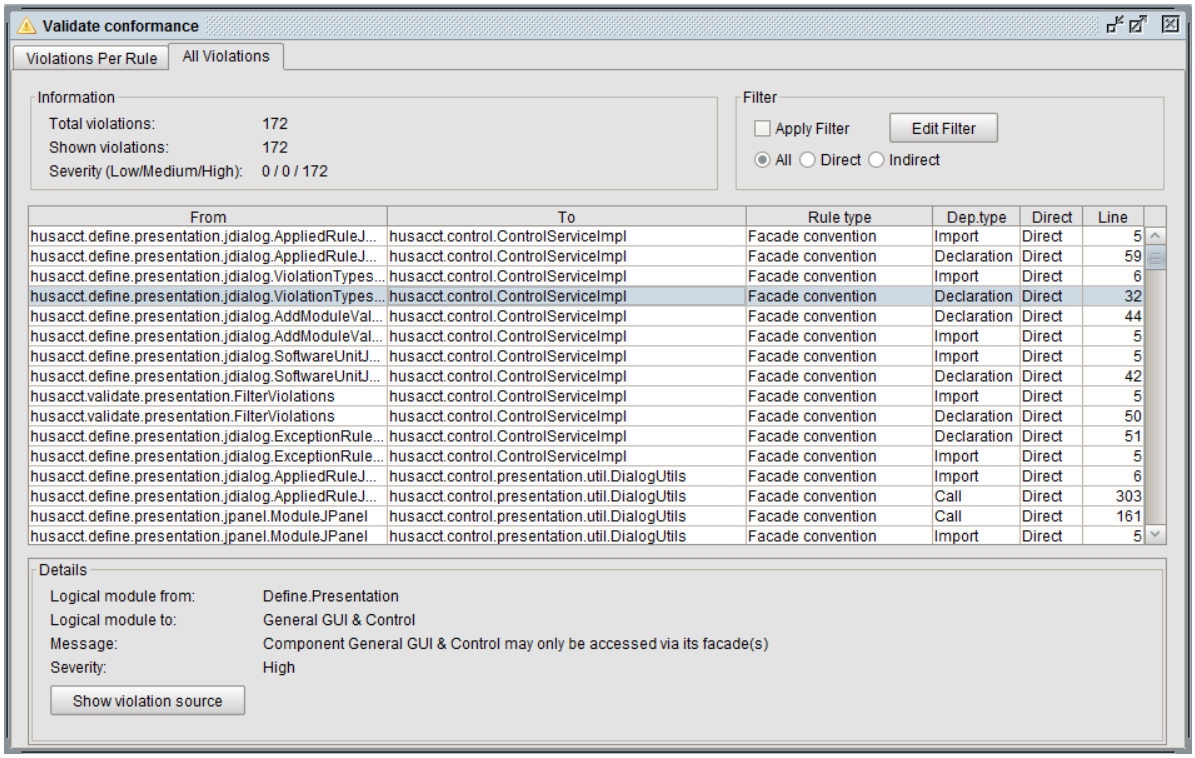
Id	Logical module from	Rule type	Logical module to	Violations
1	Analyse	Is not allowed to use	Define	1
2	Define.Presentation	Is not allowed to skip call		10
3	Define.Task	Is not allowed to back call		99
4	Define.Task	Is not allowed to skip call		24
5	General GUI & Control	Facade convention		38

From	To	Rule type	Dep.type	Direct	Line
husacct.define.presentation.jdialog.AppliedRuleJD...	husacct.define.domain.SoftwareUnitDefinition	Is not allowed to skip call	Import	Direct	7
husacct.define.presentation.moduletree.Combine...	husacct.define.domain.SoftwareUnitDefinition.Type	Is not allowed to skip call	Access	Direct	57
husacct.define.presentation.moduletree.Combine...	husacct.define.domain.SoftwareUnitDefinition.Type	Is not allowed to skip call	Declaration	Direct	57
husacct.define.presentation.moduletree.Combine...	husacct.define.domain.SoftwareUnitDefinition.Type	Is not allowed to skip call	Access	Direct	58
husacct.define.presentation.moduletree.Combine...	husacct.define.domain.SoftwareUnitDefinition.Type	Is not allowed to skip call	Import	Direct	7
husacct.define.presentation.moduletree.Combine...	husacct.define.domain.SoftwareUnitDefinition.Type	Is not allowed to skip call	Call	Direct	57
husacct.define.presentation.moduletree.Combine...	husacct.define.domain.SoftwareUnitDefinition	Is not allowed to skip call	Import	Direct	6
husacct.define.presentation.moduletree.Combine...	husacct.define.domain.SoftwareUnitDefinition	Is not allowed to skip call	Access	Direct	59
husacct.define.presentation.moduletree.Combine...	husacct.define.domain.SoftwareUnitDefinition	Is not allowed to skip call	Call	Direct	58
husacct.define.presentation.moduletree.Combine...	husacct.define.domain.SoftwareUnitDefinition	Is not allowed to skip call	Declaration	Direct	58

5.1.2 ALL VIOLATIONS

This view shows all violations. When a violation is selected, details are shown in the bottom panel.
A double click on a dependency, or a click on the button 'Show violation source', will activate the code viewer. Within the code, the violating line is highlighted in red.
Additional features are:

- Sorting: The violations can be sorted by clicking on the headers, e.g. 'Rule type'.
- Filtering: Explained below.



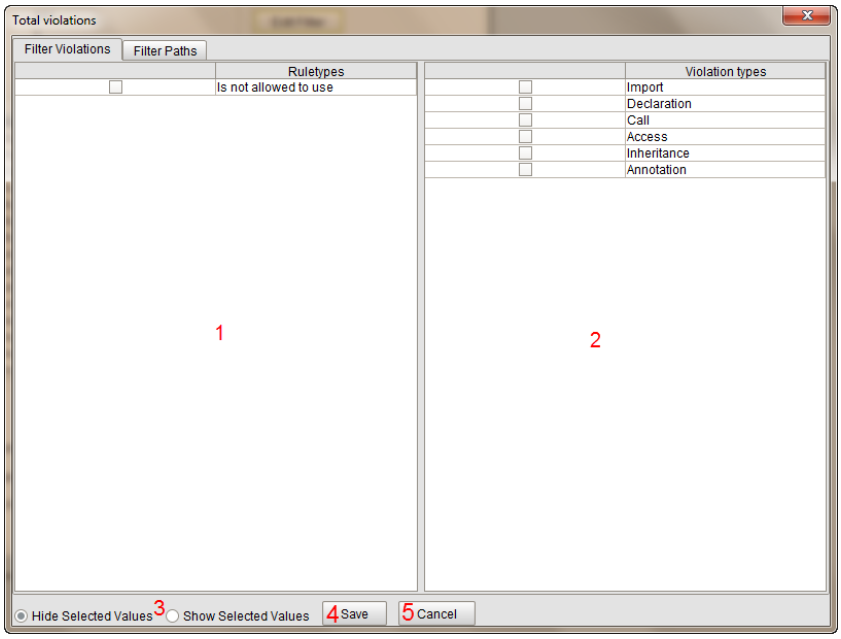
1. With the checkbox 'Apply Filter', the filter can be turned on or off.
2. When the button 'Edit Filter' is pressed the Filter dialog is opened.
3. The three radio buttons provide a filter to switch between: 1) all kinds of dependencies; 2) only direct dependencies; or 3) only indirect dependencies. This filter only works when the checkbox 'Apply Filter' is marked.

5.1.2.1 Filter dialog

This screen provides an easy to use way to filter the violations. During filtering can be chosen to show or hide the selected values. There are two tabs in this screen. In the first tab only the rule types and violation types are shown. In the second tab there is a possibility to filter on the source path of the violations.

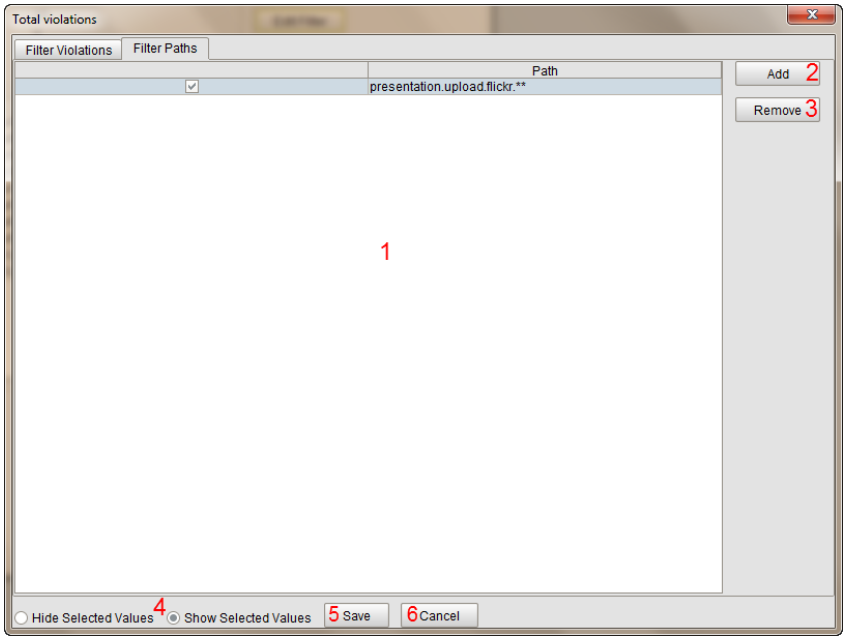
FILTER DIALOG TAB FOR FILTERING RULE- AND/OR VIOLATION TYPES

1. In this table the rule types that will be filtered can be selected.
2. In this table the violation types that will be filtered can be selected.
3. The option to choose if the filtered values must be shown or must be hidden.
4. When this button is pressed, the violations will be filtered.
5. When this button is pressed, the screen will be closed.



FILTER DIALOG TAB FOR FILTERING CLASSPATHS

1. This table shows all the paths are added to filter. The physical paths can be filtered with a regex; the possibilities will be explained in table 2.
2. When this button is pressed, the system will add an empty field to area 1.
3. When this button is pressed, the system will remove the selected row from area 1.
4. The option to choose if the filtered values must be shown or must be hidden.
5. When this button is pressed, the violations will be filtered.
6. When this button is pressed, the screen will be closed.

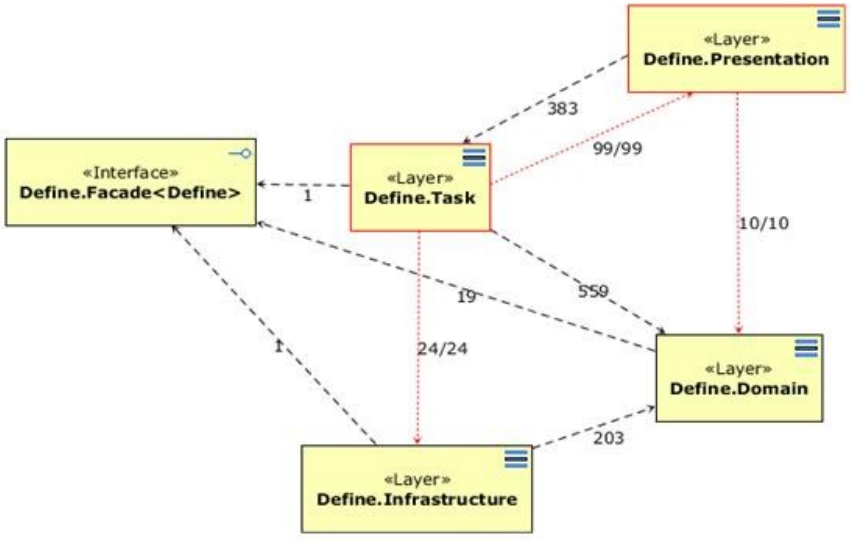


OVERVIEW OF POSSIBLE REGEXES FOR FILTERING

Your Input Example	Path Input Example	Output Example
java.St*	java.String java.StringBuffer java.String.Fake com.class.String	java.String java.StringBuffer
java.St**	java.String java.StringBuffer java.String.Fake com.class.String	java.String java.StringBuffer java.String.Fake
***.String**	java.String java.StringBuffer java.String.Fake com.class.String	java.String
****.String**	java.String java.StringBuffer java.String.Fake com.class.String	java.String com.class.String
****Stri****	java.String java.StringBuffer java.String.Fake com.class.String	java.String java.StringBuffer java.String.Fake com.class.String

5.1.3 VIOLATIONS IN DIAGRAMS

Violations may be shown in both types of diagrams, intended and implemented architecture diagrams. When the option 'Show violations' is selected, violations are made visible in the diagrams. Red dependency arrows and red modules mark violations against the rules. A red dependency arrow will show two numbers, the first represents the number of violating dependencies, while the second represents the total number of dependencies.

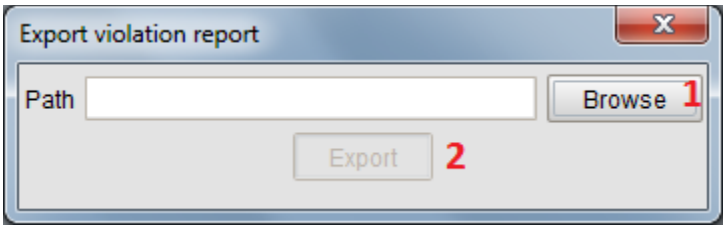


5.2 VIOLATION REPORT

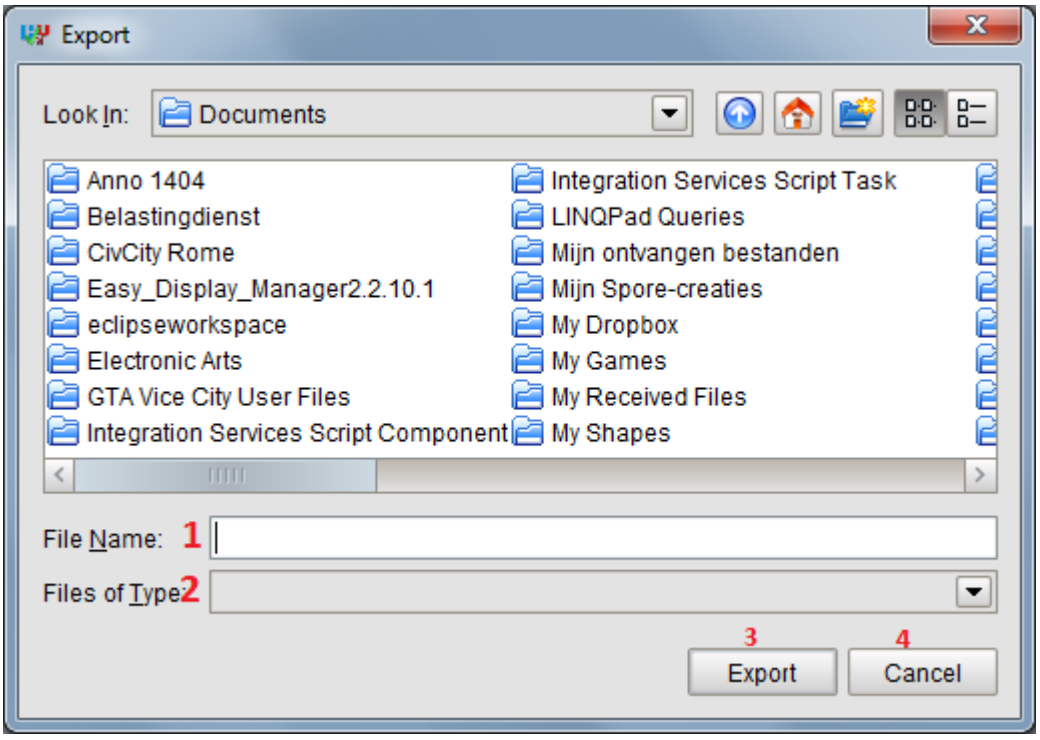
The violations can be exported to the following file types:

- XML
- HTML
- PDF

PROCEDURE



1. When this button is pressed, it opens an export dialog. (see figure 5)
2. When this button is pressed, the report will be exported.



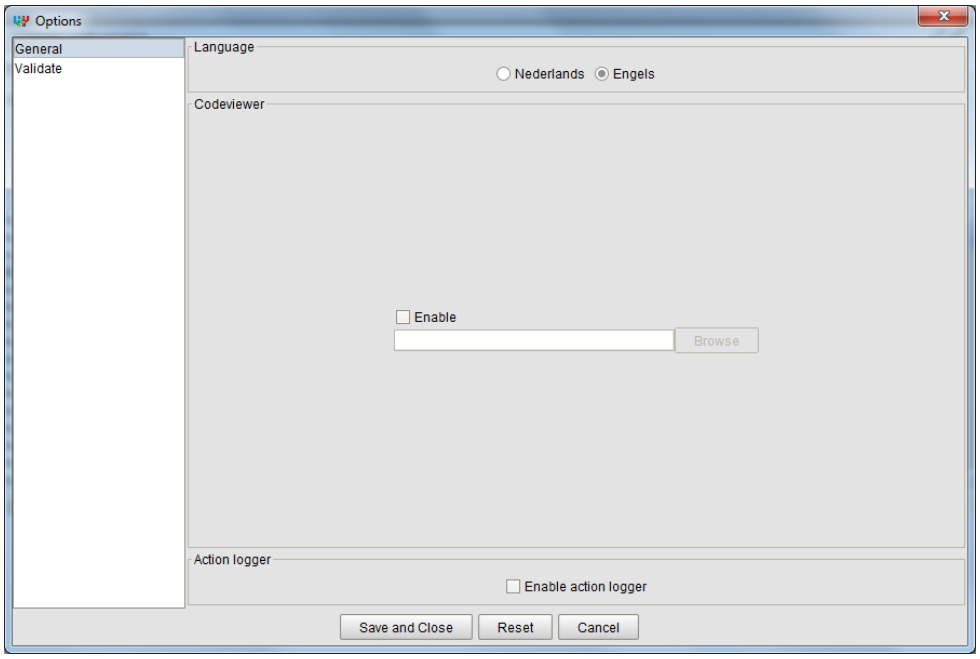
1. Select the file name and directory where the file must be saved.
2. Select the type of the file.
3. When this button is pressed, the path will be set in figure 4.
4. When this button is pressed, the screen will be closed.

6 MENU: TOOLS

6.1 OPTIONS

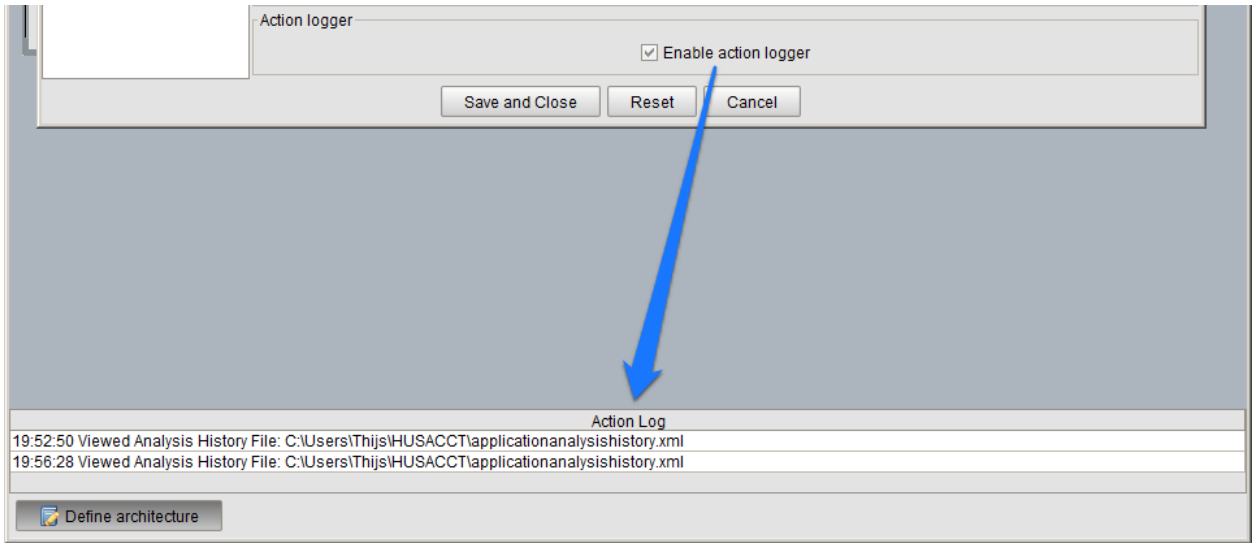
6.1.1 GENERAL

The general options allow you to: 1) select a language for the user interface of HUSACCT; 2) select a different code viewer than the build-in code viewer; 3) enable the action logger.



6.1.1.1 ACTION LOG

The action log will show actions you've taken in HUSACCT. For example, after analysis of an application a message appears in the Action Log showing the start and end time, as well as some statistics of the results. The option is disabled by default and can be enabled in Tools -> Options -> General.



6.1.2 VALIDATE - CONFIGURATION

The validate component provides also the possibility to configure the following:

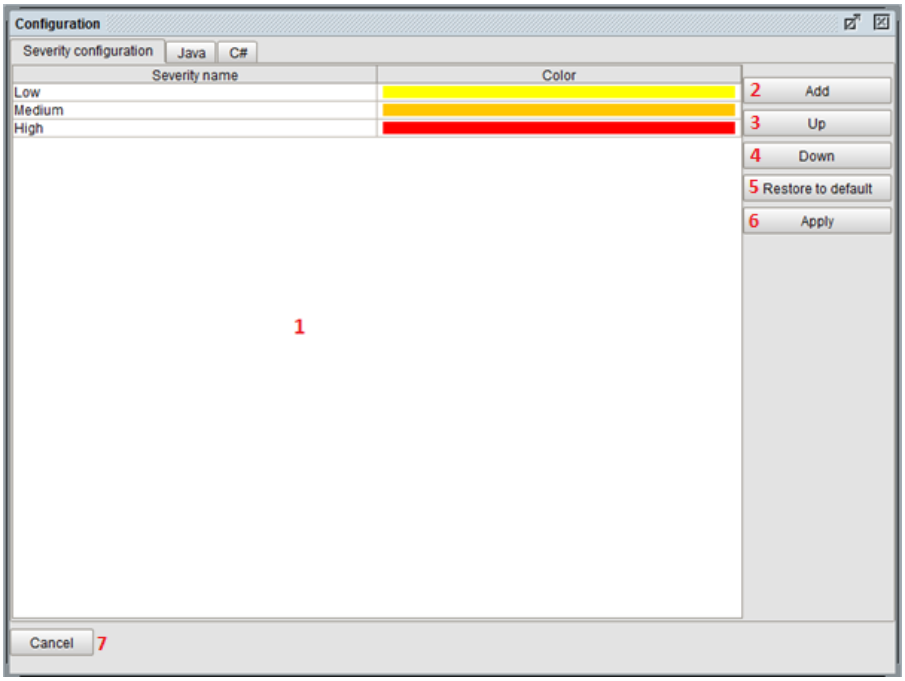
- Severities; adding new severities and change existing severities.
- Severity per rule type; change the severity for a rule.
- Severity per violation type; change the severity for a violation type.
- Active violation types; which violation types should be enabled by default in the filter when adding new rules in the define component. (For more information about filtering in rule types see the manual of the define component.)

The changes that are made are not directly reflected in the GUI. For example when severities are changed in the configuration, there must be revalidated to reflect the changes that are made in the configuration.

The screen is available once a workspace has been created. The configuration screen is available through “menu -> Validate -> Configuration”.

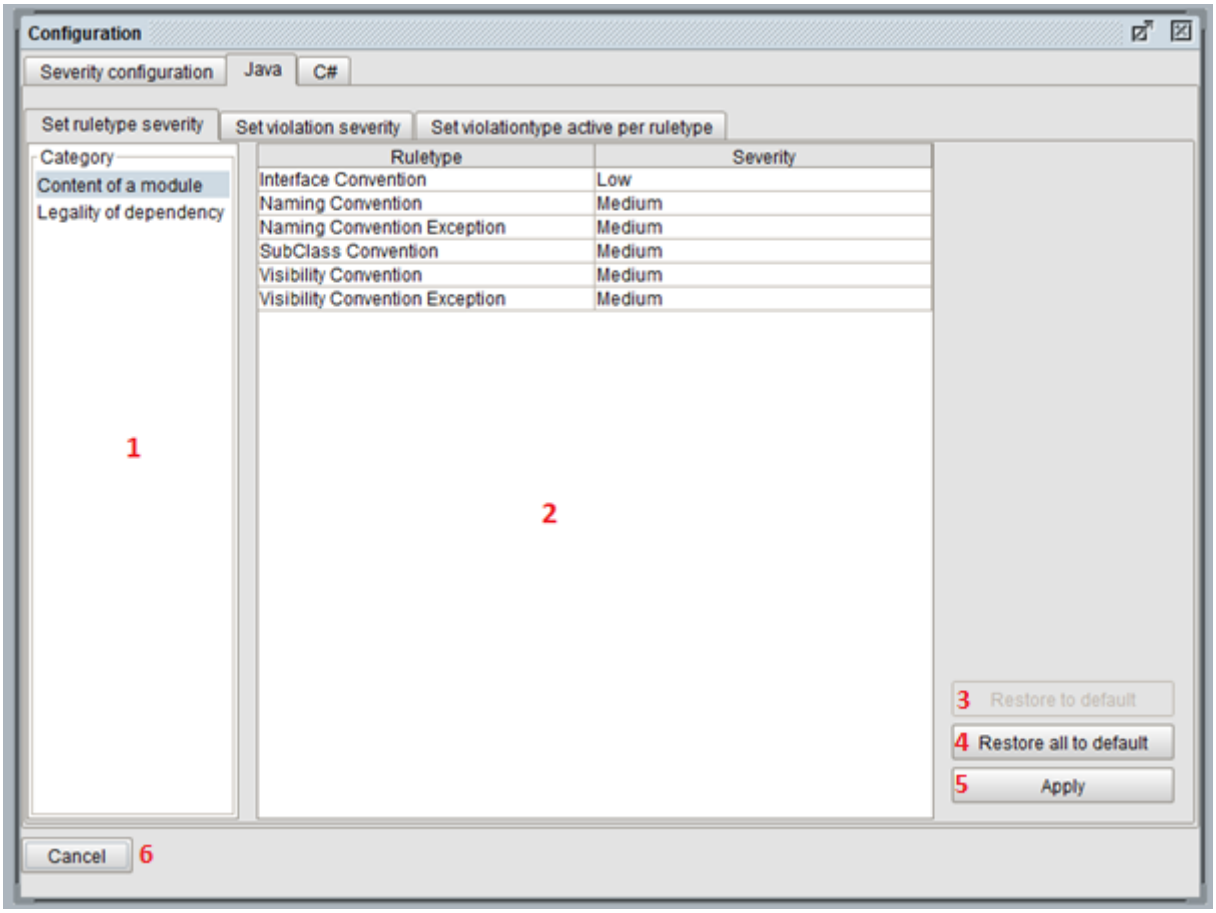
6.1.2.1 Configure severities

There are two or more tabs on the screen. The first is to create and edit severities. For every supported programming language a tab will be shown. Inside each of ‘programming language’ tab will contain three other tabs. The first two tabs are for setting the severity per rule/ violation types. The last tab is to set the active violation types.



1. This table lists all the severities. It is possible to change the name and/or color of a severity. The color can be changed by clicking on the color bar and a color picker dialog will pop up. The severities are in order from lowest to highest.
2. When this button is pressed, an empty severity will be added on the lowest place.
3. When this button is pressed, the selected severity will be moved up in the list.
4. When this button is pressed, the selected severity will be moved down in the list.
5. When this button is pressed, all the severities will be restored to the default severities.
6. When this button is pressed, all the changes in this tab will be saved.
7. When this button is pressed, the screen close without saving.

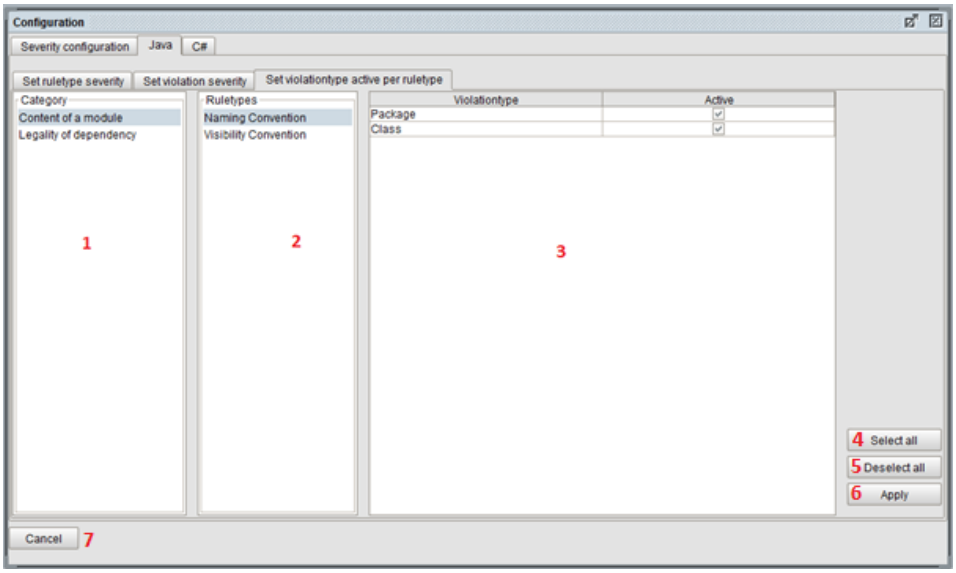
6.1.2.2 Configure severities per rule type and per violation type



This screen applies for the first two tabs.

1. List of categories.
2. Table with rule types or violation types. Changed by selecting a different category in area 1. You can change the severity by clicking on the severity name.
3. When this button is pressed, the selected rule type will be restored to its default severity level.
4. When this button is pressed, all the rule/ violation types will be restored to their default value.
5. When this button is pressed, all the changes in this tab will be saved.
6. When this button is pressed, the screen close without saving.

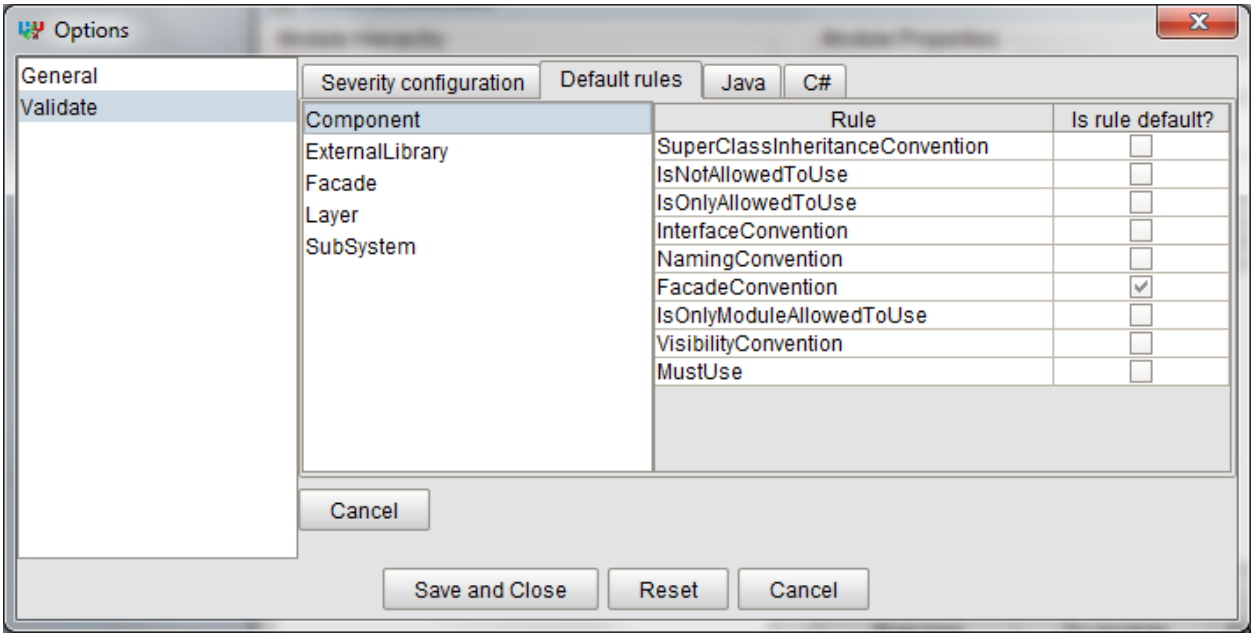
6.1.2.3 Configure active violation types



1. List of categories.
2. List of rule types. Changes when you select another category.
3. Table of active violation types. Changes when you select another rule type.
4. Select all active violation types.
5. Deselect all active violation types.
6. Save all the changes in this tab.
7. Close without saving.

6.1.2.4 Configuration of the default rules

Within the configuration, you can define what rules should be added by default, to a new component in the software architecture definition. Opening the “Default rules” tab, will lead you to a list of allowed rules per component. Select a component to set or unset the default rules in the component.



There are a few rules selected by default. These can be deselected if required. These settings are instantly saved, so no save and close are required.

7 LITERATURE

[1] Callo Arias, T.B. et al. 2011. A practice-driven systematic review of dependency analysis solutions. *Empirical Software Engineering*. 16, 5 (Mar. 2011), 544–586.

[2] Clements, P. et al. 2010. *Documenting Software Architectures: Views and Beyond*. Pearson Education.

[3] Gamma, E. et al. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education.

[4] Knodel, J. and Popescu, D. 2007. A Comparison of Static Architecture Compliance Checking Approaches. *Working IEEE/IFIP Conference on Software Architecture* (Jan. 2007), 12–21.

[5] Larman, C. 2005. *Applying UML And Patterns*. Prentice Hall PTR.

[6] Parnas, D.L. 1972. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*. 15, 12 (Dec. 1972), 1053–1058.

[7] Passos, L. et al. 2010. Static Architecture-Conformance Checking: An Illustrative Overview. *IEEE Software*. 27, 5 (2010), 82–89.

[8] Perry, D.E. and Wolf, A.L. 1992. Foundations for the Study of Software Architecture. *ACM SIGSOFT Software Engineering Notes*. 17, (1992), 40 – 52.

[9] Podgurski, A. and Clarke, L.A. 1990. A formal model of program dependences and its implications for software testing, debugging, and maintenance. *IEEE Transactions on Software Engineering*. 16, 9 (1990), 965–979.

[10] Pruijt, L. et al. 2013. Architecture Compliance Checking of Semantically Rich Modular Architectures: A Comparison of Tool Support. *2013 IEEE International Conference on Software Maintenance* (2013), 220–229.

[11] Pruijt, L. et al. 2013. On the Accuracy of Architecture Compliance Checking: Accuracy of Dependency Analysis and Violation Reporting. *21st International Conference on Program Comprehension* (San Francisco, CA, USA, 2013), 172–181.

[12] Pruijt, L. and Brinkkemper, S. 2014. A metamodel for the support of semantically rich modular architectures in the context of static architecture compliance checking. *WICSA 2014 Companion Volume* (Apr. 2014), 1–8.

[13] Pruijt, L.J. et al. 2014. HUSACCT. *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering - ASE '14* (New York, New York, USA, Sep. 2014), 851–854.

[14] Sarkar, S. et al. 2006. A method for detecting and measuring architectural layering violations in source code. *APSEC* (2006).