|  |
| --- |
| ASE TEAM 5  SYSTEM DOCUMENTATION HUSACCT – DEFINE  June 22, 2013 |

|  |
| --- |
| ROB UITHOL – SETH SNEL- LEANDER STOLK  BAYRAM KORKMAZ- REUBEN KROZENDIJK |



CONTENTS

[INTRODUCTION 4](#_Toc359869497)

[Functionality 5](#_Toc359869498)

[Use Case Model 5](#_Toc359869499)

[Goal description per Use Case 6](#_Toc359869500)

[Define logical architecture 6](#_Toc359869501)

[Define rules 6](#_Toc359869502)

[Map logical architecture to software units 6](#_Toc359869503)

[Import logical architecture 6](#_Toc359869504)

[Export logical architecture 6](#_Toc359869505)

[Import workspace 6](#_Toc359869506)

[Export workspace 6](#_Toc359869507)

[Details per use case 7](#_Toc359869508)

[Define Logical Architecture 7](#_Toc359869509)

[Define rules 8](#_Toc359869510)

[Map logical architecture to physical architecture 9](#_Toc359869511)

[References 10](#_Toc359869512)

[Define Logical Architecture 10](#_Toc359869513)

[Define rules 10](#_Toc359869514)

[Map logical architecture to software units 10](#_Toc359869515)

[Sequence diagrams 11](#_Toc359869516)

[Define logical architecture 11](#_Toc359869517)

[Define rules 12](#_Toc359869518)

[Map logical architecture to physical architecture 13](#_Toc359869519)

[Decisions and justifications 14](#_Toc359869520)

[Software partitioning 15](#_Toc359869521)

[Physical Class Diagram 16](#_Toc359869522)

[Modular architecture Define 18](#_Toc359869523)

[Top-level packages 18](#_Toc359869524)

[Domain 18](#_Toc359869525)

[Task 19](#_Toc359869526)

[Presentation 19](#_Toc359869527)

[Persistency 20](#_Toc359869528)

[Analyzer 20](#_Toc359869529)

[Responsibility Trace Table 21](#_Toc359869530)

[Subsystem specification 22](#_Toc359869531)

[Module Tree 22](#_Toc359869532)

[Classes 22](#_Toc359869533)

[Sequence Diagram 23](#_Toc359869534)

[Rule Details 23](#_Toc359869535)

[Classes 23](#_Toc359869536)

[Class diagram 24](#_Toc359869537)

[Contradictory rule conventions checker 25](#_Toc359869538)

[Classes 26](#_Toc359869539)

[Sequence Diagram 26](#_Toc359869540)

[Domain model 27](#_Toc359869541)

[Classes 27](#_Toc359869542)

[Applied Rules 28](#_Toc359869543)

[Modules 28](#_Toc359869544)

[Information model 29](#_Toc359869545)

[Domain Parser 30](#_Toc359869546)

[Sequence Diagram 30](#_Toc359869547)

[View in browser 31](#_Toc359869548)

[How-To-Add 32](#_Toc359869549)

[New Rule 32](#_Toc359869550)

[GUI 32](#_Toc359869551)

[Factory 33](#_Toc359869552)

[Testing 34](#_Toc359869553)

[Future work 35](#_Toc359869554)

[Known bug list 35](#_Toc359869555)

[Ideas for improvement 35](#_Toc359869556)

[Ideas for extension 36](#_Toc359869557)

# INTRODUCTION

The abbreviation ‘HUSACCT’ stands for Hogeschool Utrecht Software Architecture Compliance Checking Tool and is used to check the software architecture of an application. By analysing the code, you can breakdown the structure, view a graphical representation of the application and compare the analysed software to the architectural model. The tool allows you to create logical modules; create rules that apply to these modules and map the actual physical entities to the module.

This HUSACC Tool has been divided in multiple categories; every category broadly is a function, a process of the application as a whole. The category elaborated in this document is the ´Define´- component. This component is restricted to map the physical entities to the architectural model. Meaning that this is the place where you identify your modules, where you assign packages to modules and where you create rules (with exceptions) for modules.

The development of this software was done over two years with a different team per year. The initial development team of this component was:

* Bob Sanders
* Henk ter Harmsel
* Dennis van den Waardenburg
* Alex Schouls

The second and last development team of this component was:

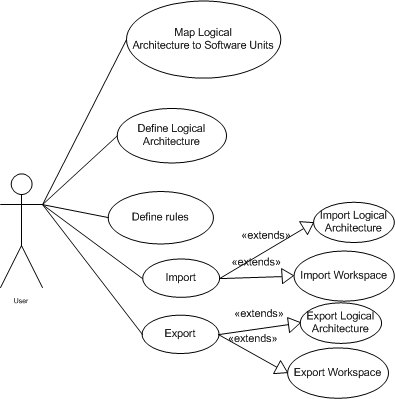
* Rob Uithol
* Seth Snel
* Leander Stolk
* Bayram Korkmaz
* Reuben Krozendijk

# Functionality

This section will elaborate on the use cases.  
There are seven architectural significant use cases:

* Define logical architecture
* Define rules
* Map logical architecture to software units
* Import logical architecture
* Export logical architecture
* Import workspace
* Export workspace

## Use Case Model



## Goal description per Use Case

### Define logical architecture

The goal of this use case is to make the user able to define a logical architecture.  
This means defining architectural components (layers, subsystems etc.)

### Define rules

This use case complements “Define logical architecture” for the goal here is to define rules on the defined components of the architecture. Some of the rules are automatically added as they are required for the architectural component.

### Map logical architecture to software units

This use case’s purpose is to map the defined architecture (See also: Define logical architecture) to the analysed software units. Note that this can only be done when an application has been analysed.

### Import logical architecture

Import logical architecture’s purpose is to import an existing architecture for future use or mapping.

### Export logical architecture

Export logical architecture’s purpose is to export the logical architecture for storage, therefor increasing the reusability of the logical architecture, since it can be used for multiple applications.

### Import workspace

Importing the workspace will import all properties of a saved project.

### Export workspace

Exporting a workspace will save all properties of a project.

## Details per use case

Not all use cases require elaboration. This part will elaborate on:

* Define logical architecture
* Define rules
* Map logical architecture to physical architecture

### Define Logical Architecture

|  |  |
| --- | --- |
| USE CASE |  |
| Number: | 1.0 |
| Version: | 2.0 |
| Writer | Team 5 - Define |
| Priority | Must |
| Use Case | Define Architecture 🡪 Define Logical Architecture |
| Actors | User |
| Summary | It’s possible to define the desired logical architecture to check the architecture. |
| Precondition | Actor has created a workspace |
| Main scenario | |  |  | | --- | --- | | **Actor actions** | **System actions** | |  | 1. The system shows the GUI to define the architecture. | | 1. Actor clicks “New module”. 2. Actor fills in the module details, and its type. 3. Actor clicks on “Save”. | 1. The system adds the module to the list. | |
| Post condition | The logical architecture is now defined and can be saved or imported. |
| Transformation rules |  |
| Constraints |  |

### 

### Define rules

|  |  |
| --- | --- |
|  |  |
| Number: | 5.0 |
| Version: | 2.0 |
| Writer | Team 5 - Define |
| Priority | Must |
| Use Case | Define Architecture 🡪 Define Rules |
| Actors | User |
| Summary | Define a rule for a specific module |
| Precondition | Modules are defined for applying the rule on |
| Main scenario | |  |  | | --- | --- | | **Actor actions** | **System actions** | |  | 1. The system shows the GUI to define the architecture. | | 1. Actor selects an already defined module. |  | | 1. Actor clicks on “Add” under the “Rules” section. |  | |  | 1. The system shows the popup screen for adding a new rule | | 1. Actor selects the rule type, the module it applies to, whether it’s enabled or not and possible exceptions. |  | | 1. Actor clicks on “Add” | 1. The system creates the rule and adds it to the list of rules | |
| Post condition | The rule is defined and applied to the specific module. |
| Transformation rules |  |
| Constraints |  |

### Map logical architecture to physical architecture

|  |  |
| --- | --- |
| USE CASE |  |
| Number: | 2.0 |
| Version: | 2.0 |
| Writer: | Team 5 - Define |
| Priority | Must |
| Use Case | Define Architecture 🡪 Map Logical Architecture to Physical Architecture |
| Actors | User |
| Summary | This use cases describes mapping of the logical defined architecture to the physical architecture of an application. |
| Precondition | An architecture has to be defined and an application has to be analysed |
| Main scenario | |  |  | | --- | --- | | **Actor actions** | **System actions** | |  | 1. The system shows the GUI to define the architecture. | | 1. Actor either defines the logical architecture or imports an existing one 2. Actor selects a module which he/she would like to map. 3. Actor clicks on “Add” under the “Software Units” section. 4. Actor selects the software unit he/she would like to map to the selected module. 5. Actor clicks on “Assign” | 1. The system shows the popup screen for assigning a software unit. 2. The system maps the logical architecture to the software units | |
| Post condition | The defined logical architecture must be used and mapped to the software units present in the application. |
| Transformation rules |  |
| Constraints |  |

## References

### Define Logical Architecture

|  |  |
| --- | --- |
| PATH | TYPE |
| Define.presentation.jpanel.ModuleJPanel | Class |
| Define.presentation.jpanel.EditModuleJPanel | Class |
| Define.presentation.jdialog.AddModuleValuesJDialog | Class |
| Define.task.DefinitionController | Class |
| Define.domain.services.ModuleDomainService | Class |
| Define.domain.SoftwareArchitecture | Class |
| Define.domain.module | Package |

### Define rules

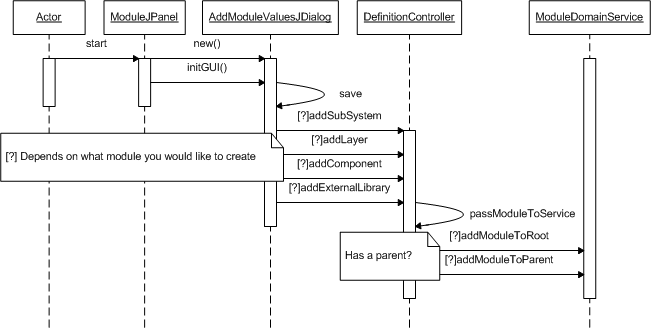
|  |  |
| --- | --- |
| PATH | TYPE |
| Define.presentation.jpanel. AppliedRulesJPanel | Class |
| Define.presentation.jpanel.ruledetails | Package |
| Define.presentation.moduleTree | Package |
| Define.presentation.jdialog.AppliedRuleJDialog | Class |
| Define.presentation.jdialog.ExceptionRuleJDialog | Class |
| Define.task.DefinitionController | Class |
| Define.task.AppliedRuleController | Class |
| Define.domain.services. AppliedRuleDomainService | Class |
| Define.domain.services.AppliedRuleExceptionDomainService | Class |
| Define.domain.SoftwareArchitecture | Class |
| Define.domain.AppliedRule | Class |

### Map logical architecture to software units

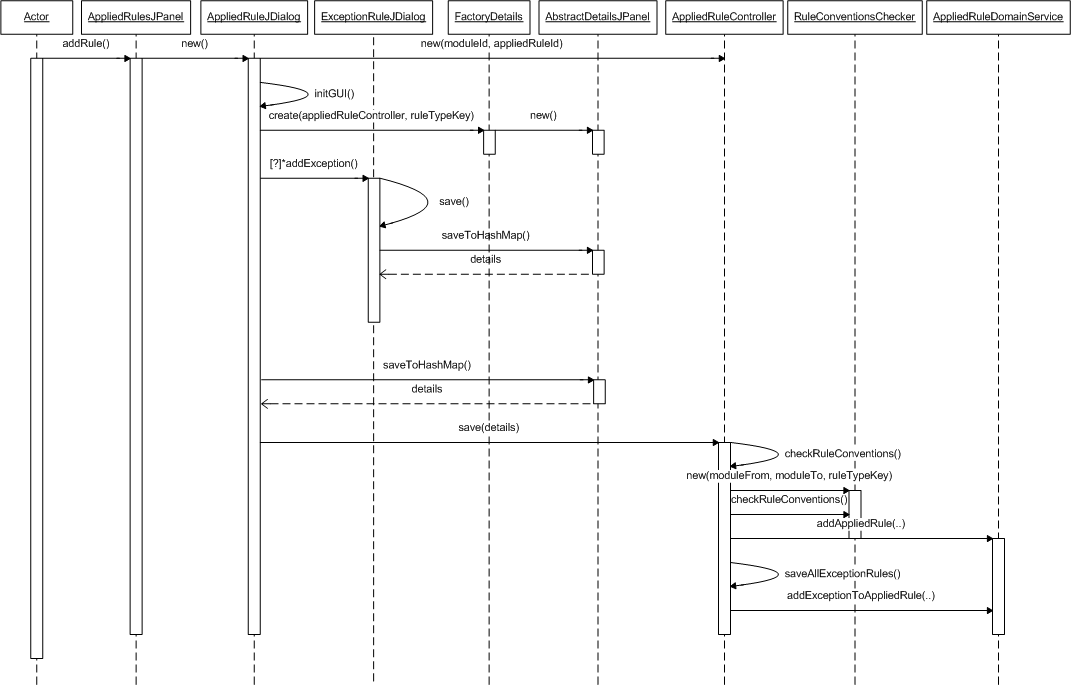
|  |  |
| --- | --- |
| PATH | TYPE |
| Define.presentation.jpanel. SoftwareUnitsJPanel | Class |
| Define.presentation.jdialog.SoftwareUnitJDialog | Class |
| Define.task.DefinitionController | Class |
| Define.task.SoftwareUnitController | Class |
| Define.domain.services.SoftwareUnitDefinitionDomainService | Class |
| Define.domain.SoftwareArchitecture | Class |
| Define.domain. SoftwareUnitDefinition | Class |

## Sequence diagrams

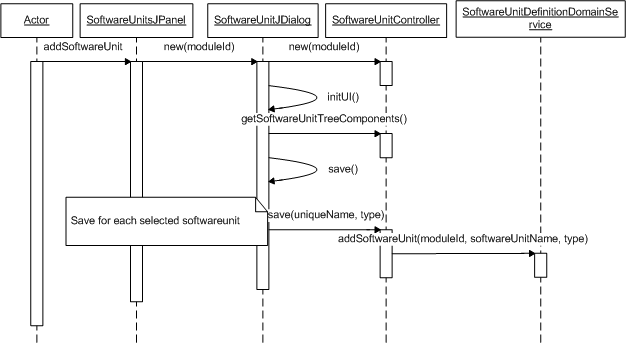
### Define logical architecture



### Define rules



### Map logical architecture to physical architecture



# Decisions and justifications

|  |  |
| --- | --- |
| Decision | Applied rule has a composition with itself |
| Where | Conceptual Domain Model, class AppliedRule |
| Reason | It is now possible to define exception rules on an applied rule making it possible to define different types of AppliedRule exceptions which can store different data depending on the RuleType. |

|  |  |
| --- | --- |
| Decision | Applied rule contains information about the violation types |
| Where | Conceptual Domain Model, class AppliedRule |
| Reason | NFR 3.2 |

|  |  |
| --- | --- |
| Decision | Separate the task from the presentation |
| Where | Package task |
| Reason | NFR 4 (And required by Team 1) |

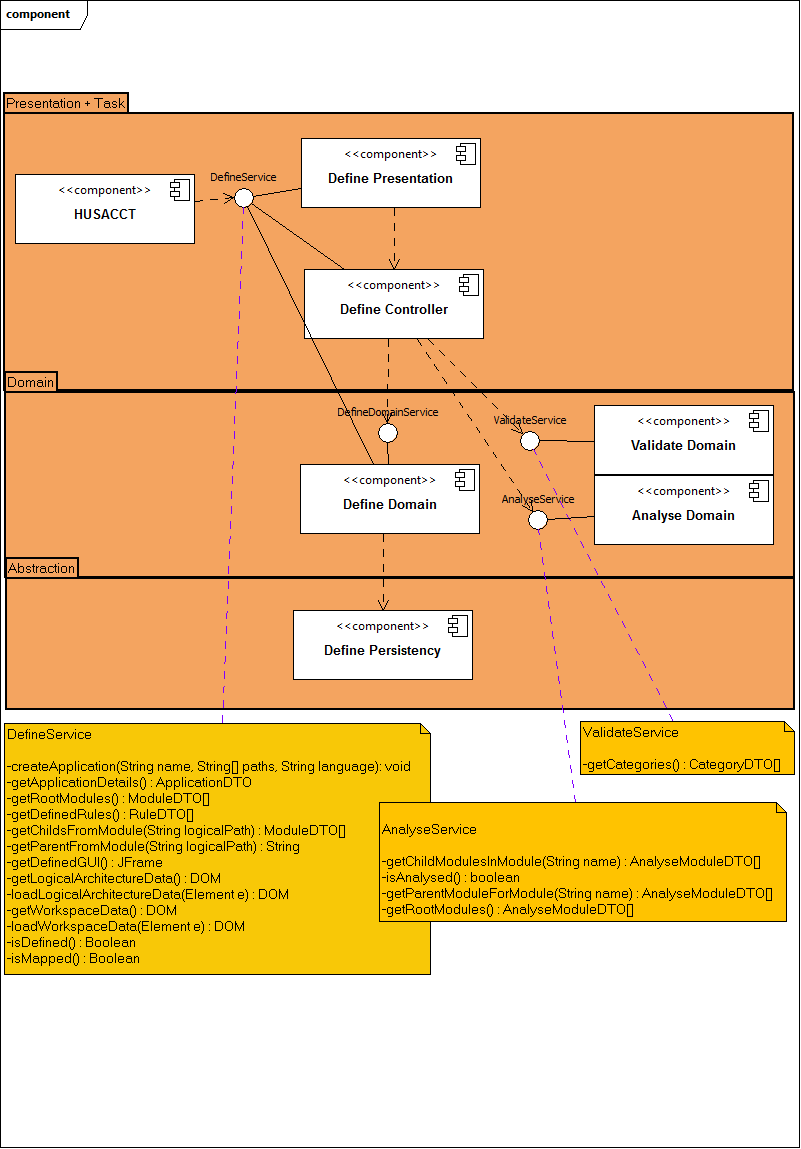
|  |  |
| --- | --- |
| Decision | Using Singleton pattern |
| Where | Package domain |
| Reason | It is better to have only one instance of Architecture Definition, and it must be accessible to clients from a well-known access point. |

|  |  |
| --- | --- |
| Decision | Using Domain Services |
| Where | Domain package |
| Reason | Low coupling between layers. |

|  |  |
| --- | --- |
| Decision | Using Property files for switching language |
| Where | Common package |
| Reason | NFR 1.1, NFR 3.5 (And required by Team 1) |

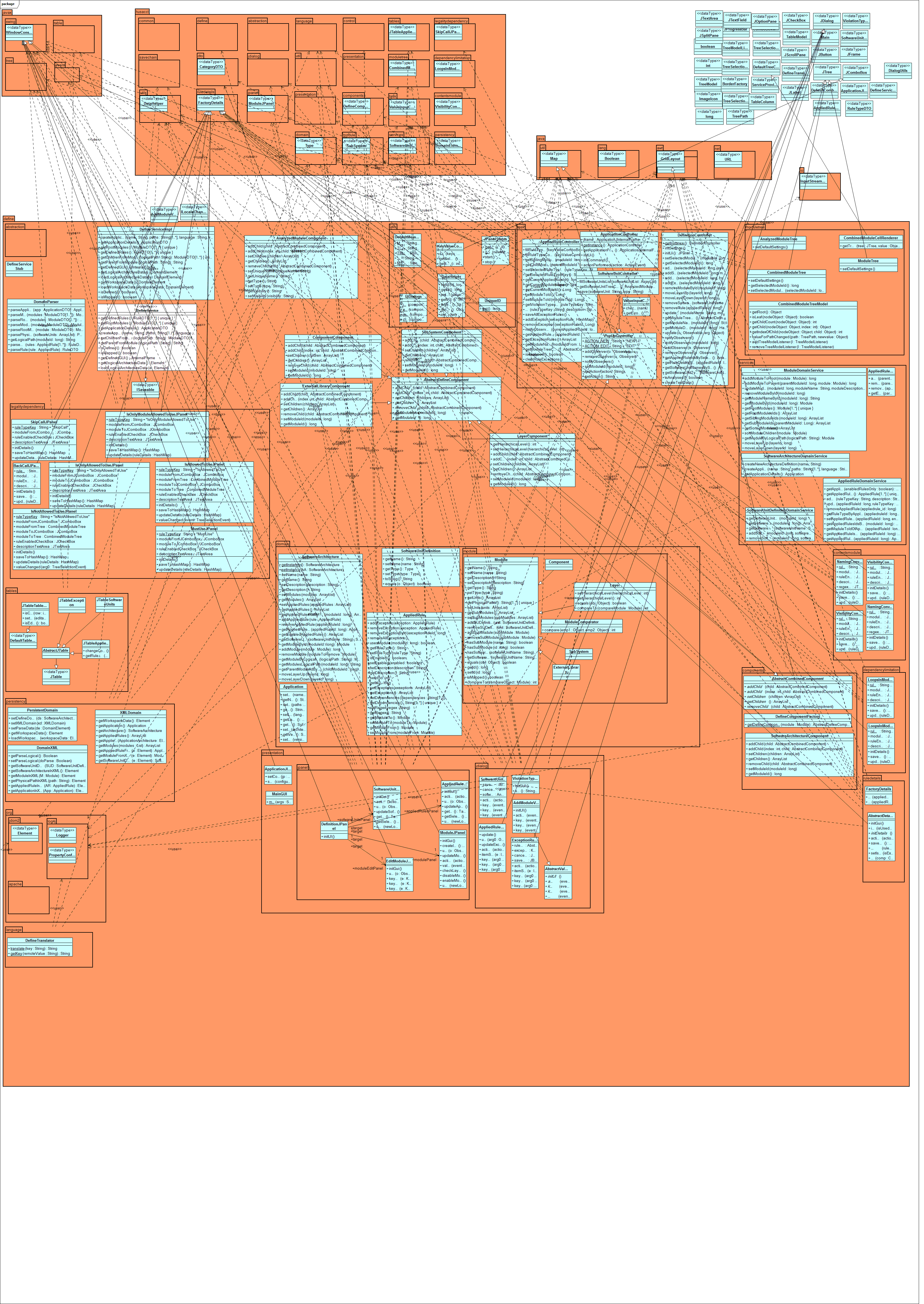
|  |  |
| --- | --- |
| Decision | HUSACCT does not support physical/ghost modules |
| Where | Domain package |
| Reason | We choose not to allow a module to be defined as a ghost module. A ghost module means it is not visible in the GUI and has a sole purpose to combine a rule and a software unit. However a ghost module is unjustifiable in the information model. So we chose to create a logical module instead when a software unit is selected. |

# Software partitioning



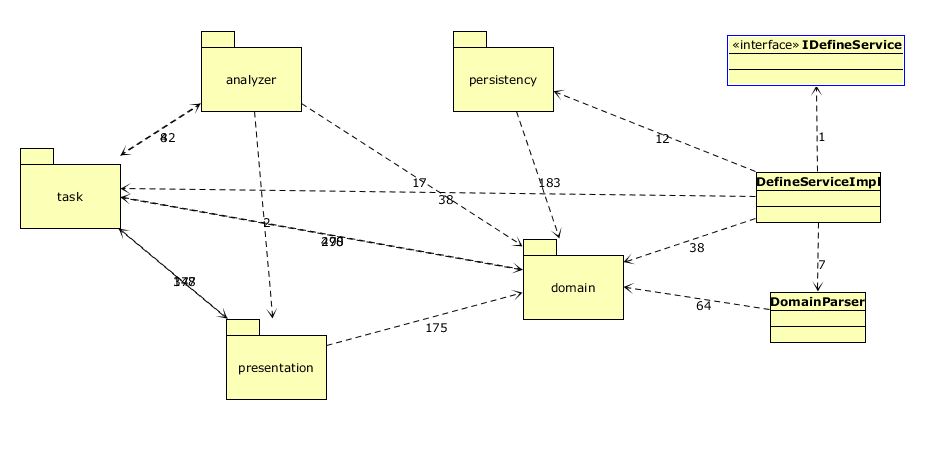
# Physical Class Diagram

The physical class diagram shows all the classes and packages used by the define component. It shows links with other packages too. The diagram has been implemented in Topcased 5.2.0.

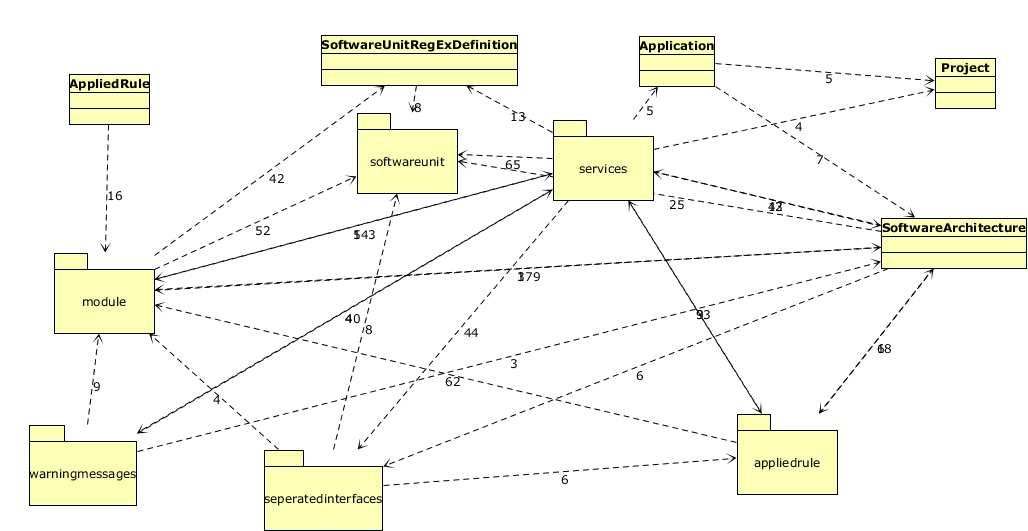


# Modular architecture Define

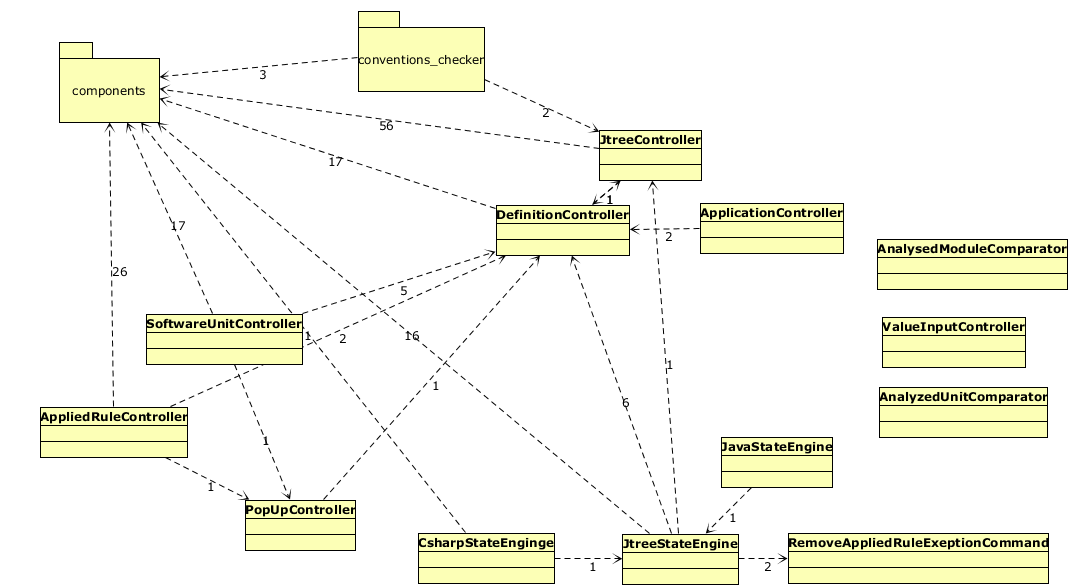
## Top-level packages

Decomposition of lower level packages

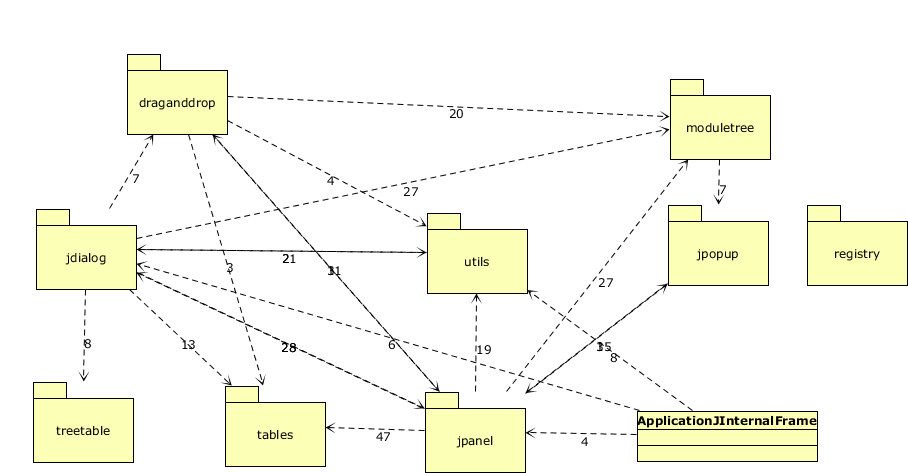
### Domain



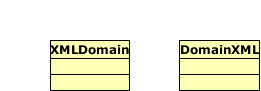
### Task



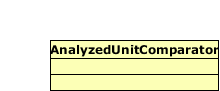
### Presentation



### Persistency



### Analyzer



# Responsibility Trace Table

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Type of Logic  → | User Interface | | | Task Specific | | | Domain Generic | | | Infrastructure Abstraction | | Infrastructure | |
| Responsibility →  Software Layer ↓ | UI Construction | Event Capturing | Event Processing | Task Control | Task State Maintenance | TS Operation | DG Service Control | DG Data Transfer | DG Operation | Application Platform Abstraction | Infrastructure Application Abstraction | Application Platform Service | Infrastructure Application Service |
| Define |  |  |  | x |  | x |  |  |  |  |  |  |  |
| Analyzer |  |  |  |  |  | x |  |  |  |  |  |  |  |
| Domain |  |  |  |  |  |  | x | x | x |  |  |  |  |
| Persistency |  |  |  |  |  |  |  |  |  | x | x | x | x |
| Presentation | x | x | x |  |  |  |  |  |  |  |  |  |  |
| Task |  |  |  | x | x | x |  |  |  |  |  |  |  |

# Subsystem specification

Some subsystems in the define component are very large and/or hard to understand. For each of this subsystems in the define component we tried to created 1 or more of the following information to get an insight into the subsystem and to explain certain difficulties related to the subsystem.

* Classes
* Sequence diagrams showing important mechanisms
* Other diagrams explaining the subsystem

## Module Tree

This subsystem is very important. It is used 3 times for different purposes. Also the creation of this subsystem is quite difficult to understand if you first see the code. Therefore we did create a sequence diagram to show how the process operates. The module tree itself is placed in the presentation layer. The (view) components used placed in the task layer because the components are filled in the AppliedRuleController which is also placed in the task layer.

There are 3 types of trees. All trees use the same CellRenderer and TreeModel. The TreeModel causes the children of the tree to be displayed in the right manner. The CellRenderer takes care of what has to be displayed. In this tree that will be an icon for each different Component, and its name or path. The 3 tree types are:

* AnalyzedModuleTree

*This shows the path of analyzed software units in a tree.*

* ModuleTree

*Shows all defined modules in a tree.*

* CombineModuleTree

*Combining above-mentioned tree’s*

### Classes

|  |
| --- |
| HUSACCT.DEFINE.PRESENTAION.MODULETREE |
| AnalyzedModuleTree |
| ModuleTree |
| CombinedModuleTree |
| CombinedModuleCellRenderer |
| CombinedModuleTreeModel |

|  |
| --- |
| HUSACCT.DEFINE.TASKS.COMPONENTS |
| AbstractCombinedComponent |
| DefineComponentFactory |
| AbstractDefineComponent |
| ComponentComponent |
| DefineComponentFactory |
| ExternalLibraryComponent |
| LayerComponent |
| SoftwareArchitectureComponent |
| SubSystemComponent |

### Sequence Diagram



## Rule Details

RuleDetails is another very important subsystem. The mechanism is difficult to understand because it has a lot of classes. The UML diagram below shows that it is just a factory pattern.

### Classes

|  |
| --- |
| HUSACCT.DEFINE.PRESENTAION.JPANEL |
| AppliedRulesJPanel |
| DefinitionJPanel |
| EditModuleJPanel |
| ModuleJPanel |
| SoftwareUnitsJPanel |
| *PACKAGE - ruledetails* |
| AbstractDetailsJPanel |
| FactoryDetails |
| *PACKAGE - ruledetails.components* |
| AbstractPanelComponent |
| DescriptionPanelComponent |
| EnabledPanelComponent |
| ModuleFromPanelComponent |
| ModuleToPanelComponent |
| RegexPanelComponent |
| *PACKAGE - ruledetails.contentsmodule* |
| InterfaceConventionJPanel |
| NamingConventionExceptionJPanel |
| NamingConventionJPanel |
| SubClassConventionJPanel |
| VisibilityConventionExceptionJPanel |
| VisibilityConventionJPanel |
| *PACKAGE - ruledetails.dependencylimitation* |
| CyclesBetweenModulesExceptionJPanel |
| CyclesBetweenModulesJPanel |
| *PACKAGE - ruledetails.legalitydependency* |
| BackCallJPanel |
| IsAllowedToUseJPanel |
| IsNotAllowedToUseJPanel |
| IsOnlyAllowedToUseJPanel |
| IsOnlyModuleAllowedToUseJPanel |
| MustUseJPanel |
| SkipCallJPanel |

### Class diagram



## Contradictory rule conventions checker

When applying rules on defined modules, it is easily done to define contradictory rules. This subsystem checks the selected from, and sometimes to, module for existing contradictory rules. First, it checks which rule type is selected and second, on the basis of this selection, the correct checks are performed. The following rules are contradictory:

|  |  |
| --- | --- |
| RULE | FORBIDDEN WHEN THE 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. |

### Classes

|  |
| --- |
| HUSACCT.DEFINE.DOMAIN.APPLIEDRULE.PROPERTYRULES |
| FacadeConventionRule |
| InterfaceConventionRule |
| NamingConventionRule |
| SuperClassInheritanceConventionRule |
| VisibilityConventionRule |
| Task.conventions\_checker.LayerCheckerHelper |

## Domain model

### Classes

|  |
| --- |
| HUSACCT.DEFINE.DOMAIN |
| Application |
| Project |
| SoftwareArchitecture |
| SoftwareUnitRegExDefinition |

|  |
| --- |
| HUSACCT.DEFINE.DOMAIN.MODULE |
| Component |
| ExternalLibrary |
| Layer |
| Module |
| ModuleComparator |
| SubSystem |

|  |
| --- |
| HUSACCT.DEFINE.APPLIEDRULE |
| Propertyrules/ |
| Relationrules/ |
| AppliedRuleFactory |
| AppliedRuleStrategy |

### Applied Rules

As shown above, the Applied Rule package has two folders, a factory [[1]](#footnote-1)and a strategy [[2]](#footnote-2)class. The folders contain the actual ruletype classes. Each rule type has their own validations. Within the application, you will work with AppliedRuleStrategies. You can make a new rule by using the ‘createRule(String ruleType)’ method in the factory.

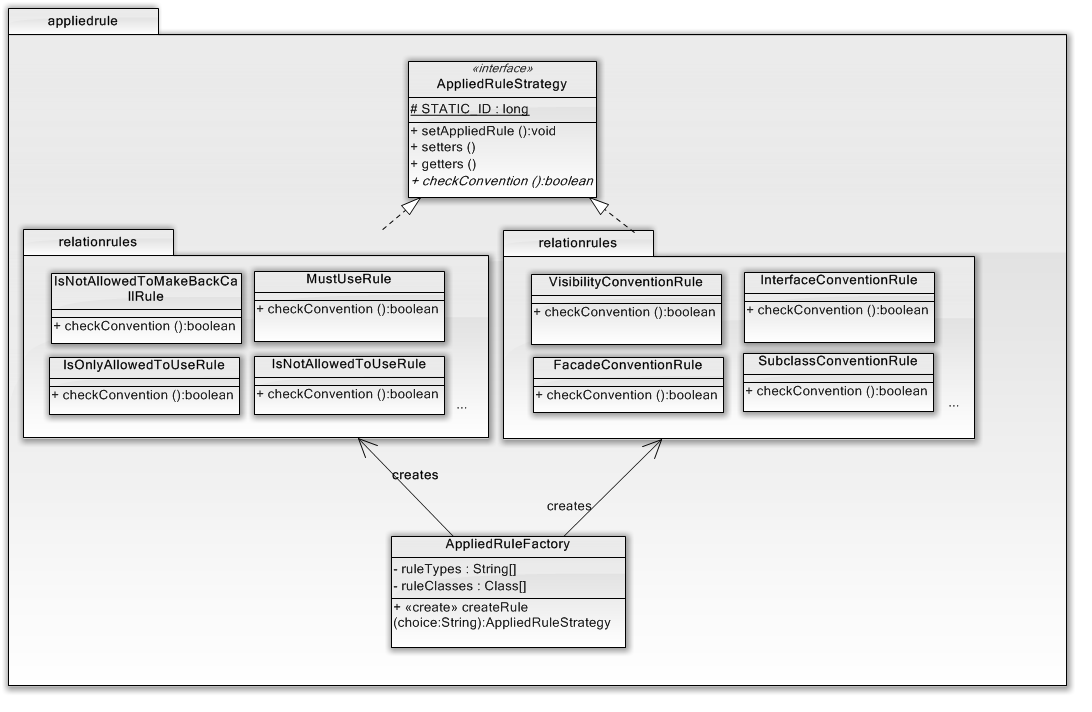
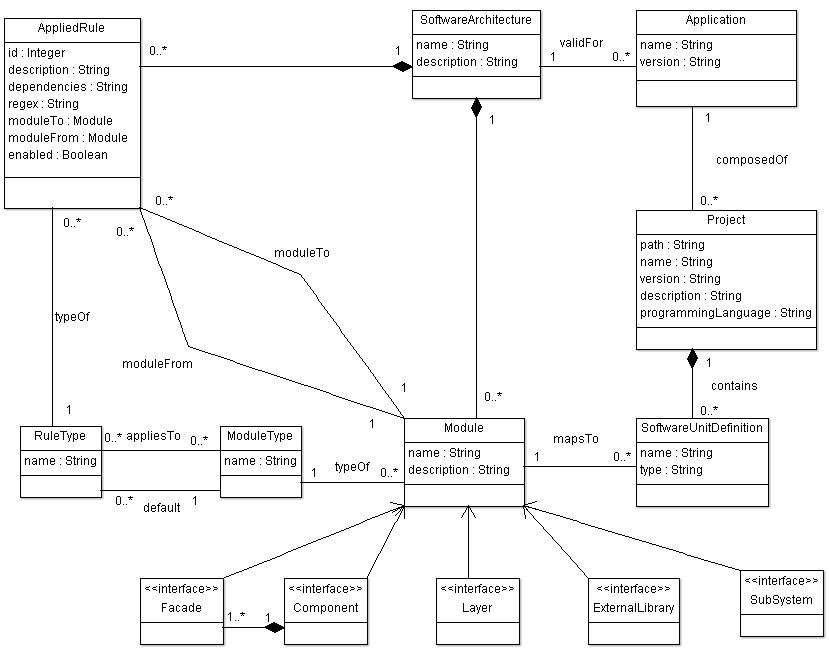


FIGURE 1

### Modules

Modules have the same structure as Applied Rules. There is an abstract ModuleStrategy, a factory that creates instances of the modules, and a folder containing all the module types.

### Information model



## Domain Parser

The domain parser is the part of define that transforms the domain objects to Data Transfer Objects. The DTO’s are placed in a shared package named common, so each component is allowed to use them. This makes it easy to transfer data between components.

In the domain parser you will find has a method to parse all units, which calls a method to parse a single object to its DTO.

### Sequence Diagram



## View in browser

In the main GUI of the define component, there is a button with the label “View in Browser”. This button creates an HTML-page with a jQuery TreeTable[[3]](#footnote-3). In the Templates Folder of Common (common/resources/templates/template.html) you can find a template.html file that is used to create the page. At the moment of writing this document, it contains two variables (‘%SUMMARY%’ and ‘%CONTENT%’) which will be replaced in ReportToHTML.java (define/presentation/utils/ReportToHTML.java). The jQuery and jQuery TreeTable are both included in the HTML-file.

# How-To-Add

## New Rule

To add a new rule, you’ll need to add a couple of things. Firstly, Validate should implement the rule, after this, the Define component can do his work. In the domain layer there’s a Applied Rule Package. This package has a folder with property rules, a folder with relation rules, an Applied Rule Strategy class and a Applied Rule Factory.

To add a new rule a couple of things need to be changed. To start with the basics, the new rule needs to be added in the Validate Component. Once it has been added there it becomes selectable in the dropdown box at the add rule GUI. However this new rule still has no GUI of its own. To do this we need to create a new class in one of the packages in the ruledetails package which is located here: *hussact.define.presentation.jpane.ruledetails*.

### GUI

To create a GUI for the new rule you must make a new class which needs to extend the AbstractDetailsJPanel . Once this is done, you will need to add GUI component. These GUI components, located *hussact.define.presentation.jpane.ruledetails.components,* are simple JPanels which have their own representation and logic to validate their data.

To add these components to the GUI of your new rule you will need to do so as follows:

1. Add a public attribute of the component you wish to add.

**public** **class** TestRuleJPanel **extends** AbstractDetailsJPanel {

**public** EnabledPanelComponent enabledPanelComponent;

1. Initialize the new component in the *InitGUI* method. Modify the GridBagConstraints so it aligns with the rest of your components.

@Override  
**public** **void** initDetails() {  
 enabledPanelComponent = **new** EnabledPanelComponent ();  
 **this**.add(enabledPanelComponent, **new** GridBagConstraints(0, 0, 2, 1, 0.0, 0.0,   
 GridBagConstraints.*FIRST\_LINE\_START*, GridBagConstraints.*NONE*, **new** Insets(0,   
 0, 0, 0), 0, 0));  
**}**

1. Complement the following methods with your added components.

@Override  
**public** **boolean** hasValidData() {  
 **boolean** hasValidData = **true**;  
 hasValidData = hasValidData && enabledPanelComponent.hasValidData(); **return** hasValidData;  
}  
@Override  
**public** **void** updateDetails(HashMap<String, Object> ruleDetails) {  
 **super**.updateDetails(ruleDetails);  
 enabledPanelComponent.update(ruleDetails.get("enabled"));  
}

@Override  
**public** HashMap<String, Object> saveToHashMap() {  
 HashMap<String, Object> ruleDetails = **super**.saveToHashMap();  
   
 ruleDetails.put("enabled", (Boolean) **this**.enabledPanelComponent.getValue());  
 **return** ruleDetails;  
}

Once you followed these steps for all the components you need to define this new rule you have a new GUI for this rule.

### Factory

You are not done yet. Once you added the new GUI you must still add it to the Factory.  
The factory class is located here: husacct.define.presentation.jpanel.ruledetails. FactoryDetails.

To assign the GUI you just made to the new rule you will need to do a couple of things.

First you will need to add a public static attribute to the GUI you just made.  
The value of this attribute should be the same as the ruleTypeKey of your new rule.

**public static final String** *ruleTypeKey* = *”YourRuleTypeKey”*;

Once you have added this attribute you can now modify the FactoryDetails class to return this GUI when this rule is selected. The first thing to do is to add a private attribute of the new GUI.

**private** TestRuleJPanel testRuleJPanel;

After you have done this you need to add the GUI in de create and get method of the factory.  
To do this add the follow piece of code to the *create* method.

**else** **if** (ruleTypeKey.equals(TestRuleJPanel.*ruleTypeKey*)){  
 testRuleJPanel = **new** TestRulelJPanel(appliedRuleController) **return** testRuleJPanel;   
 }

And add this piece of code to the *get* method.

**else** **if** (ruleTypeKey.equals(TestRulelJPanel.*ruleTypeKey*)){  
 **if** (testRuleJPanel == **null**) {  
 testRuleJPanel = **new** TestRulelJPanel (appliedRuleController);  
 }  
 **return** testRuleJPanel;  
}

Congratulations, you have now successfully added a new rule.

# Testing

Testing the application GUI wise in such way that functionality is assured needs to be done by an human hand. This means that all functionality, defined as use cases, needs to be tested as black box tests. The tests are to be confirmed by GUI response.

Tests are not defined for GUI actions and usage. The main reason for this is that the GUI changed during the whole development process and that we’ve tested the GUI throughout the development process. Some of the ideas for improvements that we’ve came up with are listed in the next chapter *Future Work*.

Code, apart from the GUI, has also been tested throughout the development process. This means that the code does have some bugs (see the *Known bug list*). It is however possible to test the define service. These tests are defined in *DefineServiceTests.java*. The tests are written as JUnit tests and can be run individually. Each of the test methods represents a feature in the Define Service.

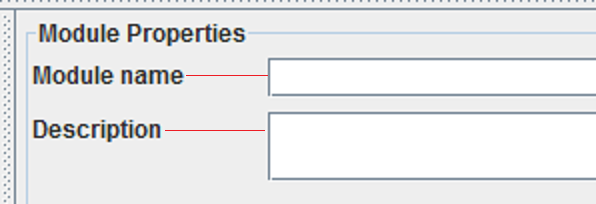
# Future work

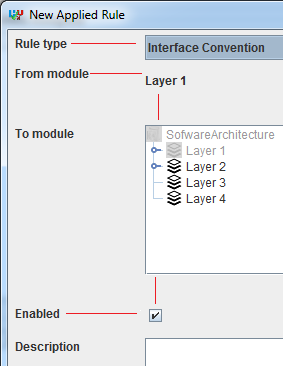
## Known bug list

* Exceptions on rules are not checked if conflicting with other rules/exceptions
* In certain cases the Validate component delivers empty results
* The user of HUSACCT can’t start the validation process until the most upper module is mapped

## Ideas for improvement

* Align GUI
  + A lot of components in the Define GUI are not properly aligned. Especially the module edit panel and the applied rule dialogs. Examples:

1)

2)

* Code quality
  + A lot can be done on code quality, but then again this can always be improved.

## Ideas for extension

* Application-Project-Software Unit
  + Currently in the domain there is an Application class which has paths to the location of the application. Furthermore all the software units are falling under one application. It is the intention that the application gets a list of projects. A project is a part of the application that is mapped to one path. This would mean that a project has a name, a path and all the software units falling under this project.
  + This also means that the Software Unit GUI needs to be adjusted to reflex the right changes. The Software Unit GUI now puts everything under one root and this need to be changed to a root for each project.
* Map software units on module via regular expressions

This is mostly realized but buggy when combined with regular mapping.

1. The Factory pattern is part of the creational patterns, *http://www.oodesign.com/factory-pattern.html ,* no author or date is mentioned, acquired at May 2013 [↑](#footnote-ref-1)
2. The Strategy pattern is part of the Behavioral patterns, *http://www.oodesign.com/strategy-pattern.html,* no author or date is mentioned, acquired at May 2013 [↑](#footnote-ref-2)
3. jQuery TreeTable http://ludo.cubicphuse.nl/jquery-treetable/ [↑](#footnote-ref-3)