|  |
| --- |
| HUSACCT Tool |
| HU Architecture Compliance Tool  22-02-2012 |
| Proof of concept Analyse Component |

Team 3

Erik Blanken

Asim Asimijazbutt

Rens Groenveld

Tim Muller

Team 6

Thijmen Verkerk

Thomas Schmidt

Martin van Haeften

Mittchel van Vliet

# Table of Contents

[Table of Contents 1](#_Toc322956495)

[1. Introduction 2](#_Toc322956496)

[2. Use Cases 3](#_Toc322956497)

[3. Analyse Application 4](#_Toc322956498)

[3.1. Sequence diagram ‘Analyse Application’ at Service Level 4](#_Toc322956499)

[3.2. Sequence diagram ‘Analyse Application’ in the general JavaAnalyser 5](#_Toc322956500)

[4. Get Dependencies 6](#_Toc322956501)

# Introduction

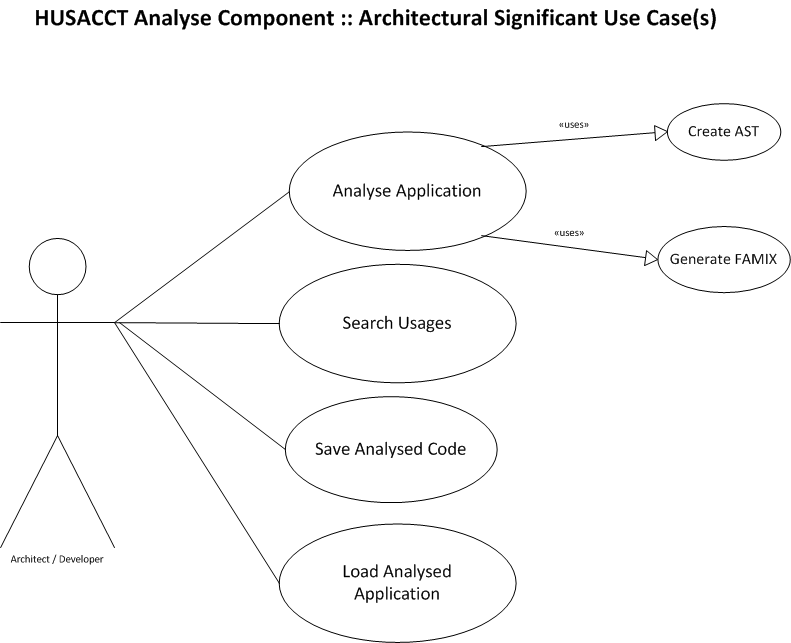
The HUSACCT-tool stands for HU Software Architecture Compliance Checking Tool. The tool is to be designed and developed, in order to enable developers and architects to compare the defined architecture with the actual developed code-base.

Based on a research about the existing SACCT-tools, this new SACCT-tool is to be designed, developed and delivered. The new tool must be able to check almost every dependency that can exist in a software-product. The tool should also be designed for new programming-languages to integrate easily.

The contents of this document, will show how the interactions between modules will be in the analyse-compontent of the HUSACCT-tool.

# Use Cases

To get an overview of all the use cases that will be implemented in the HUSACCT analyse-component, figure 1.1. shows an overview of all use cases.



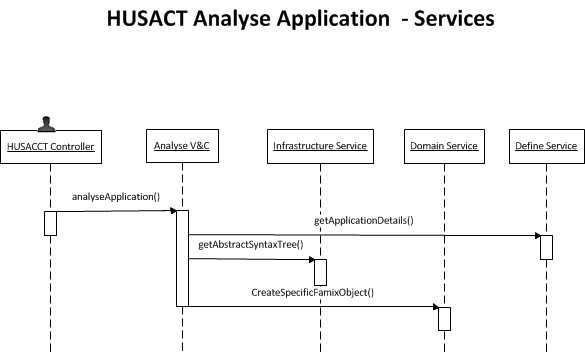
*Figure 1.1. Use cases if the Analyze-component of the HUSACCT-tool*

# Analyse Application

The following chapter will explain how the use case *Analyse Application* is to be implemented in the HUSACCT-tool.

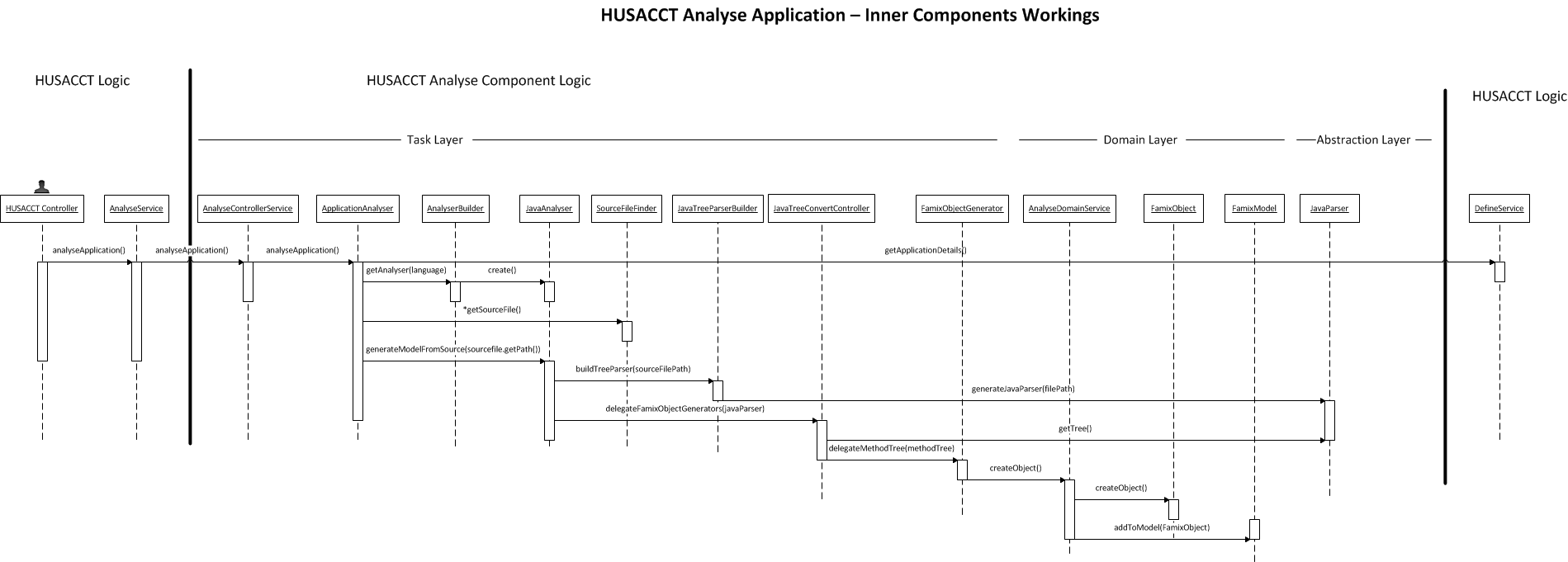
### Sequence diagram ‘*Analyse* Application’ at Service Level

*Figure 2.1. Analyze Application Sequence diagram at top-level*



### Sequence diagram ‘*Analyse Application’* in the general JavaAnalyser

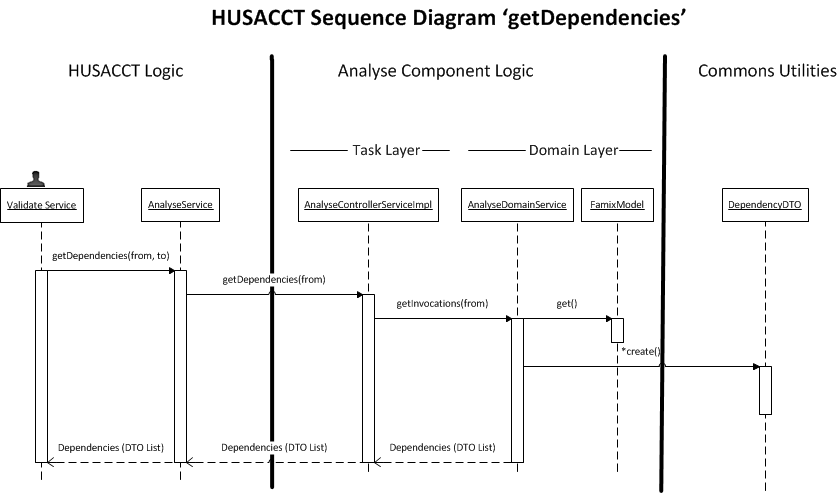
In order to make the application expandable for new languages, an generic mapper-component and a builder-class will be inserted to enable developers to implement functionalities in different manors.



*Figure 2.3. Analyse Application on class level*

# Get Dependencies

The use case *get dependencies* will be a commonly used functionality by other components, especially the component that will actually filter the violations. This makes the use case very important. The abstract overview of the implementation of this use case, is drawn in figure 3.1.



*Figure 3.1. Abstract overview of the implementation of the use case get dependencies*