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

1. 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.

2. 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.

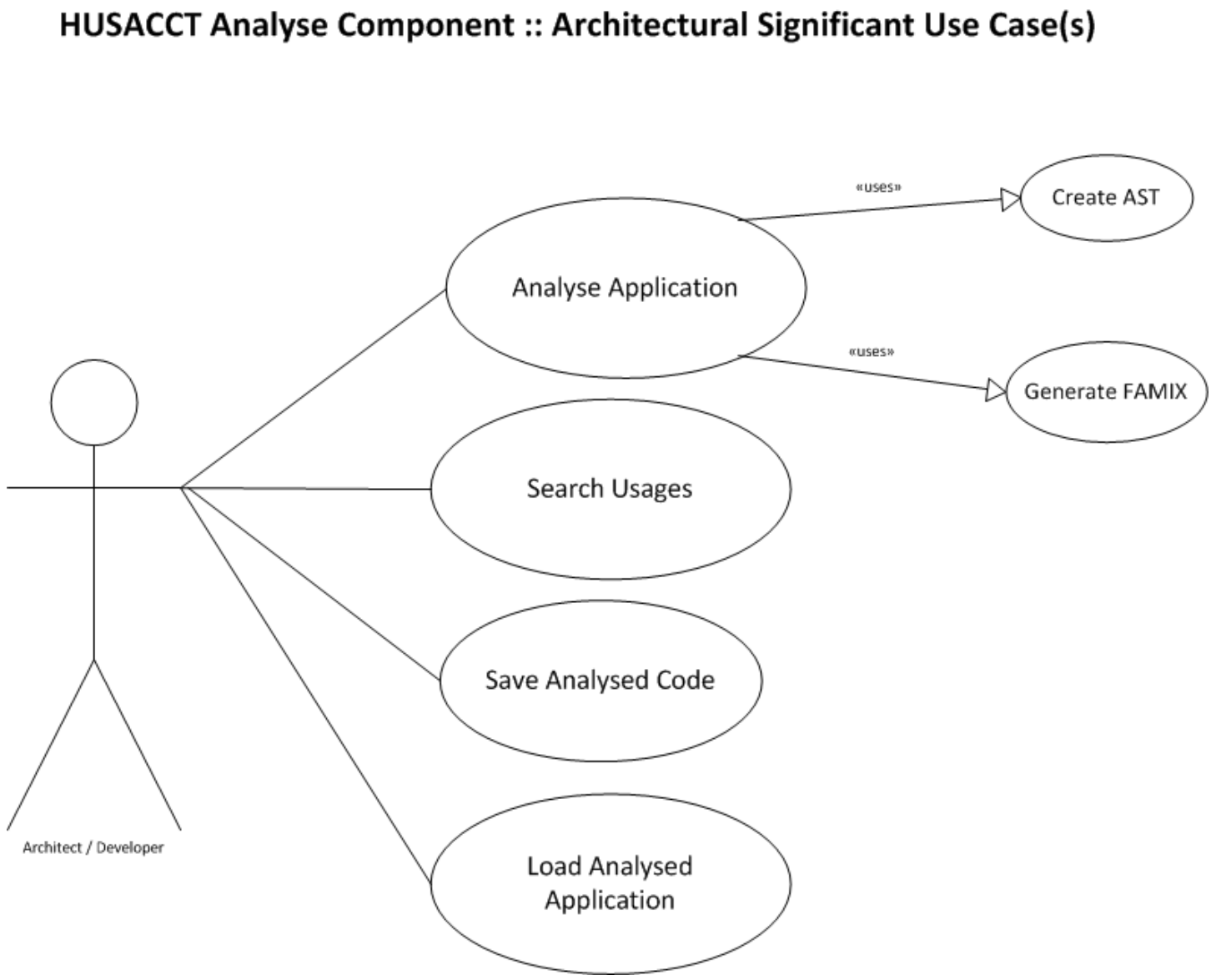

Figure 1.1. Use cases if the Analyze-component of the HUSACCT-tool

3. Analyse Application

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

3.1. Sequence diagram '*Analyse* Application' at Component Level

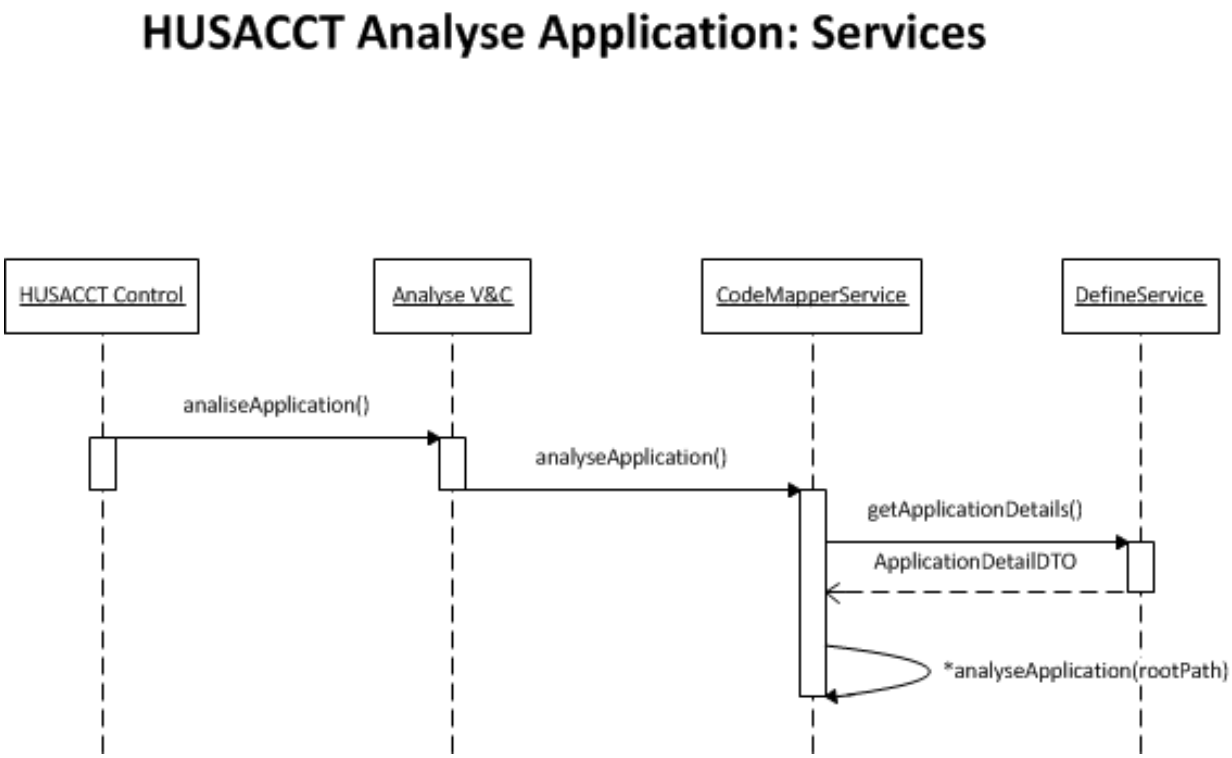

Figure 2.1. Analyze Application Sequence diagram at top-level

3.2. Sequence diagram '*Analyse* Application' for Analyse V&C

The 'Analyse V&C' will only serve as service that will trigger the analyse-service to start analyzing an application. Figure 2.2. shows how this will work.

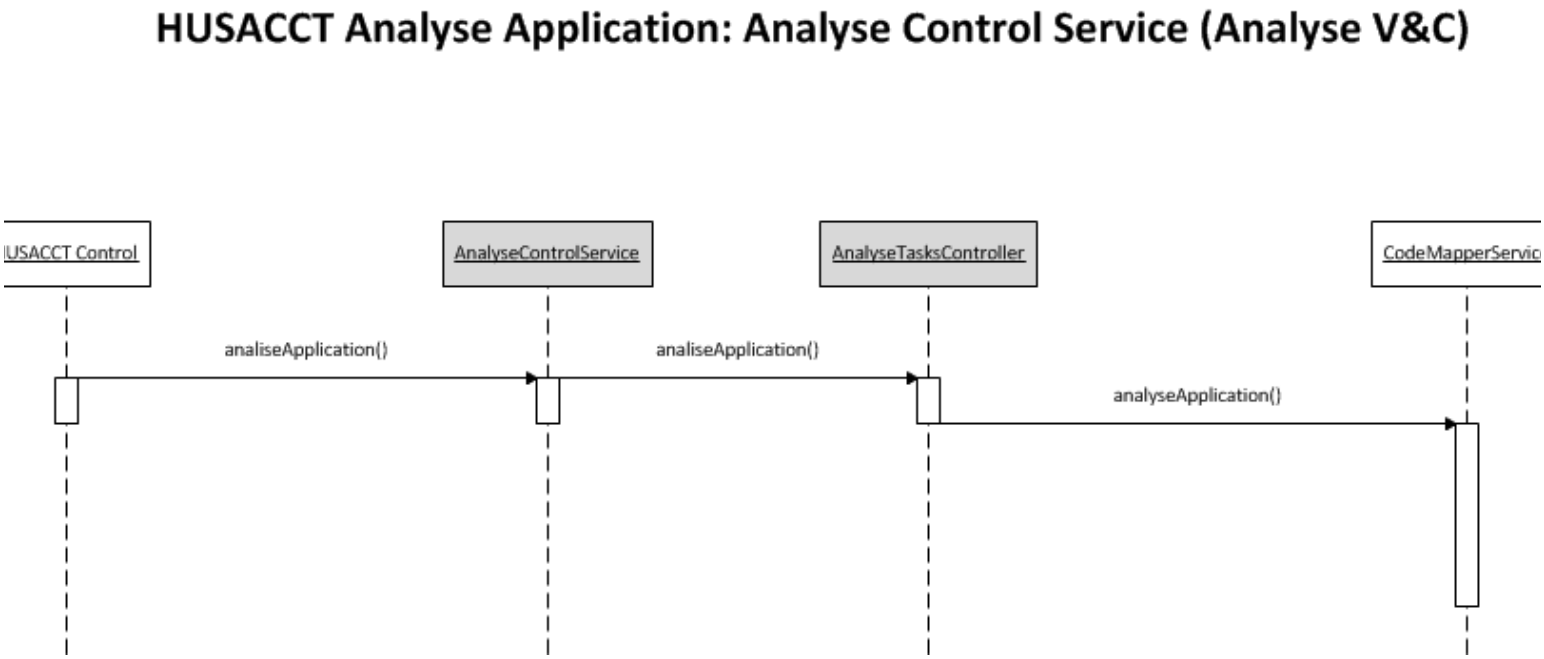

Figure 2.2. Analyze Application sent through bu the AnalyseControlService. (Service of the Analyse V&C Component)

3.3. Sequence diagram '*Analyze Application*' in the general CodeMapper

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.

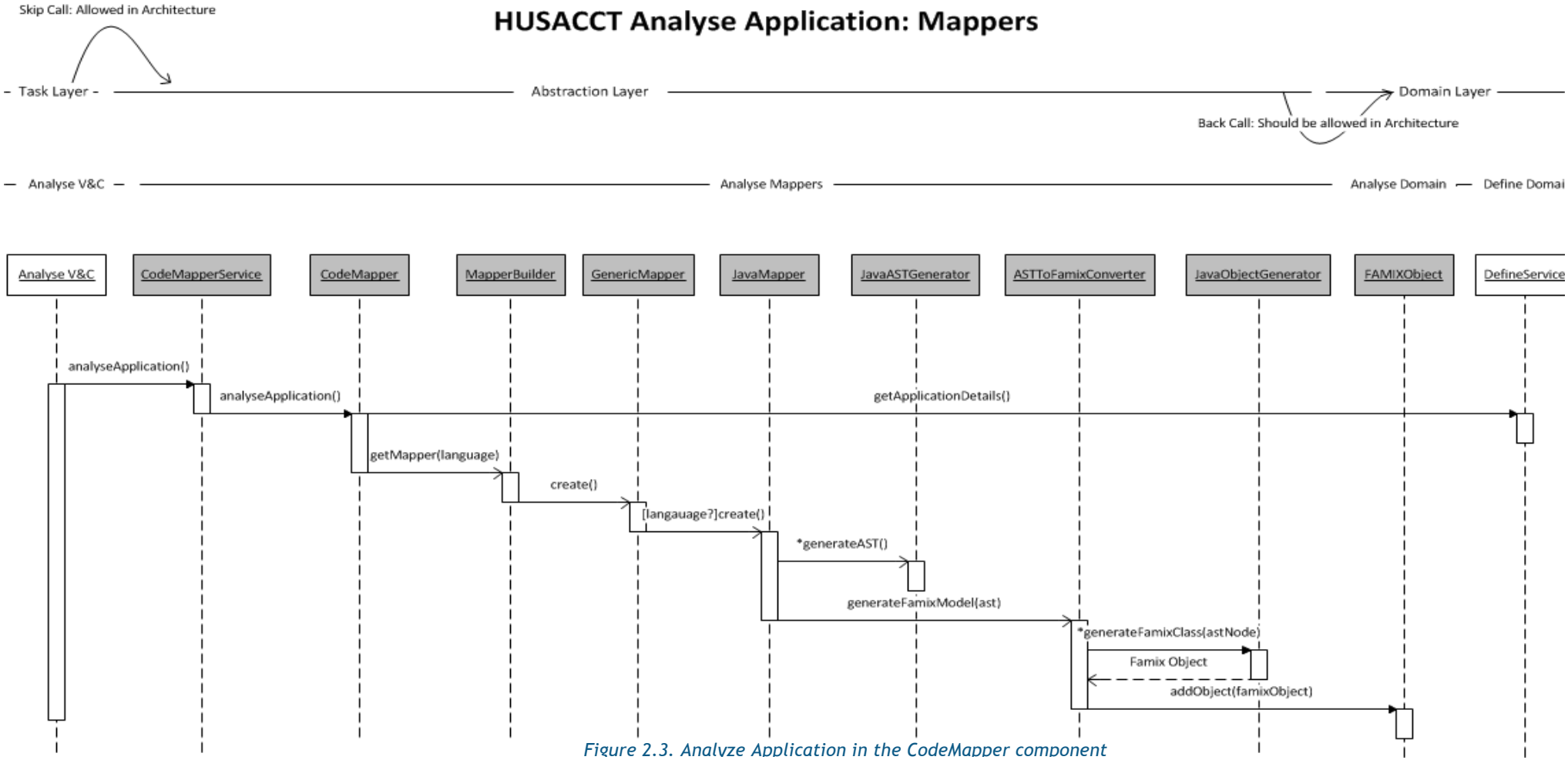

Figure 2.3. Analyze Application in the CodeMapper component

4. Search Usages

The use case *Search Usages* 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.

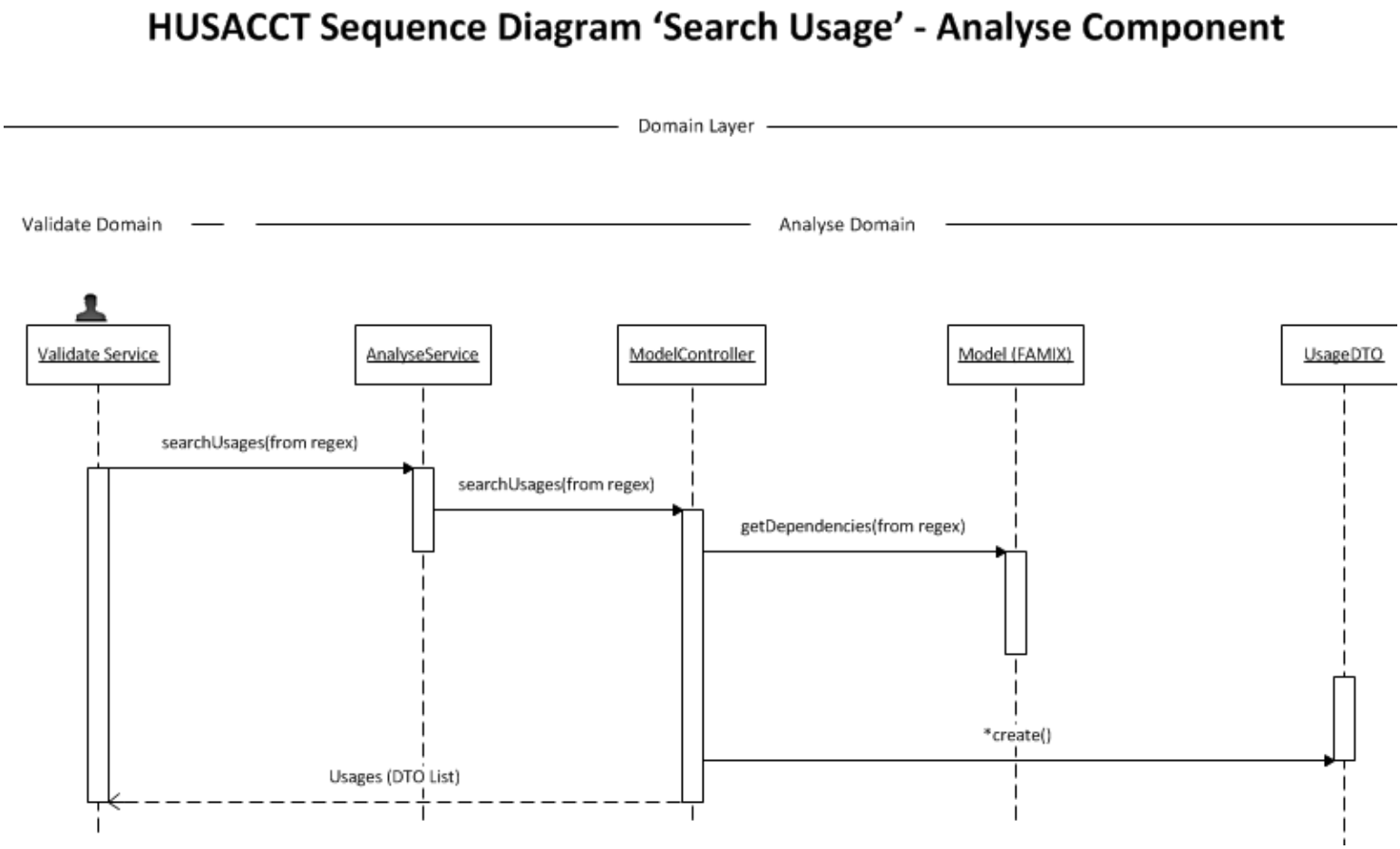

Figure 3.1. Abstract overview of the implementation of the use case Search Usage

5. Save Analysed Code

An important thing of the analyse-component is the persistency of models that were generated, based on codebases. This in order to keep performance as high as possible, because analysing large application over and over again, will be much slower than just loading the model from a location.

An overview of the implementation of this use case, is drawn in figure 4.1.

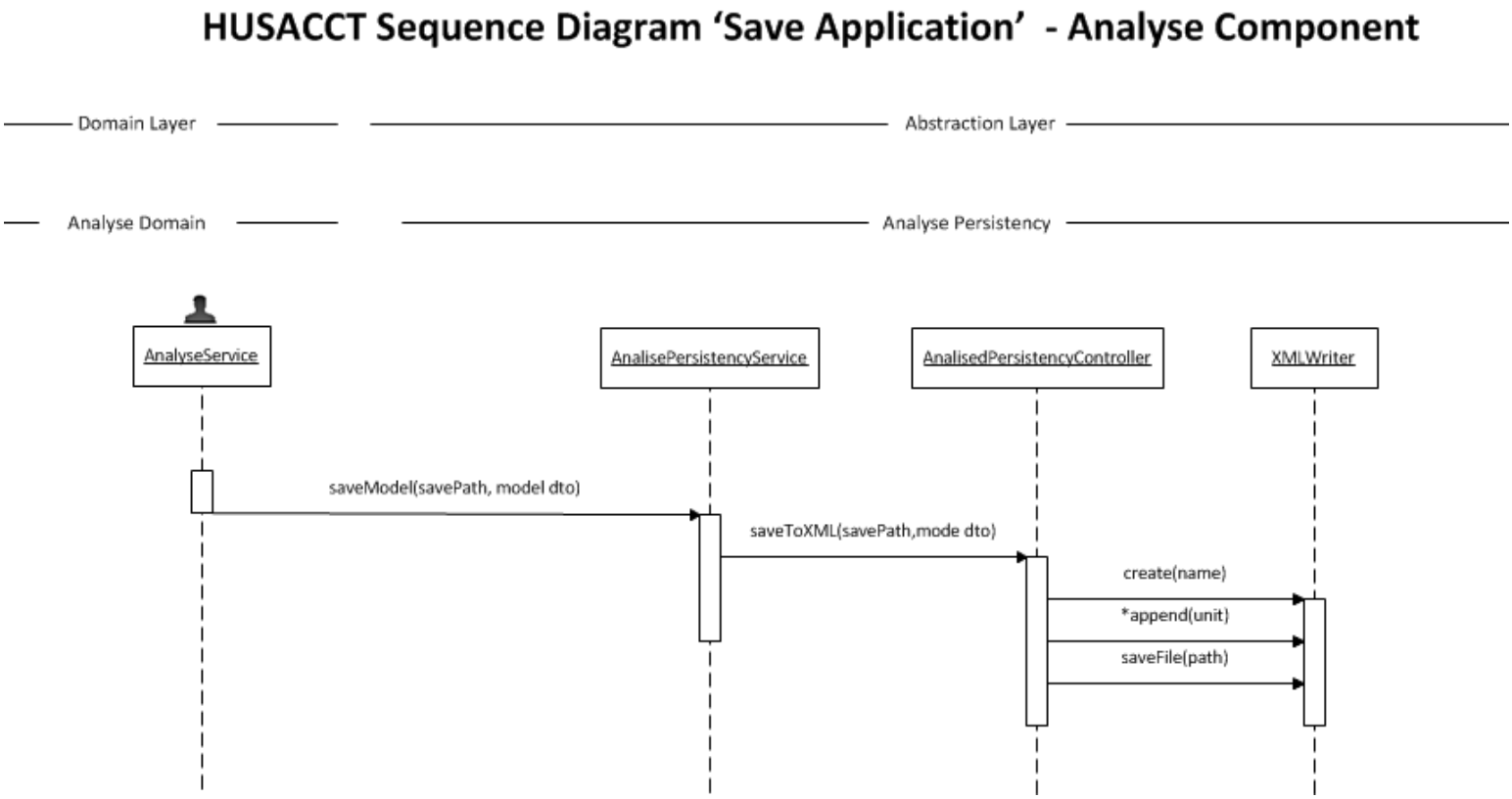

Figure 4.1. Abstract overview of the implementation of the use case Save Analysed Code

6. Open Analysed Code

To enable users to load in a model which was already created because the code was analysed earlier, the application needs to implement this functionality. This is performed in the use case *Open Analysed Code*. Figure 5.1. gives an overview of the implementation of this use case.

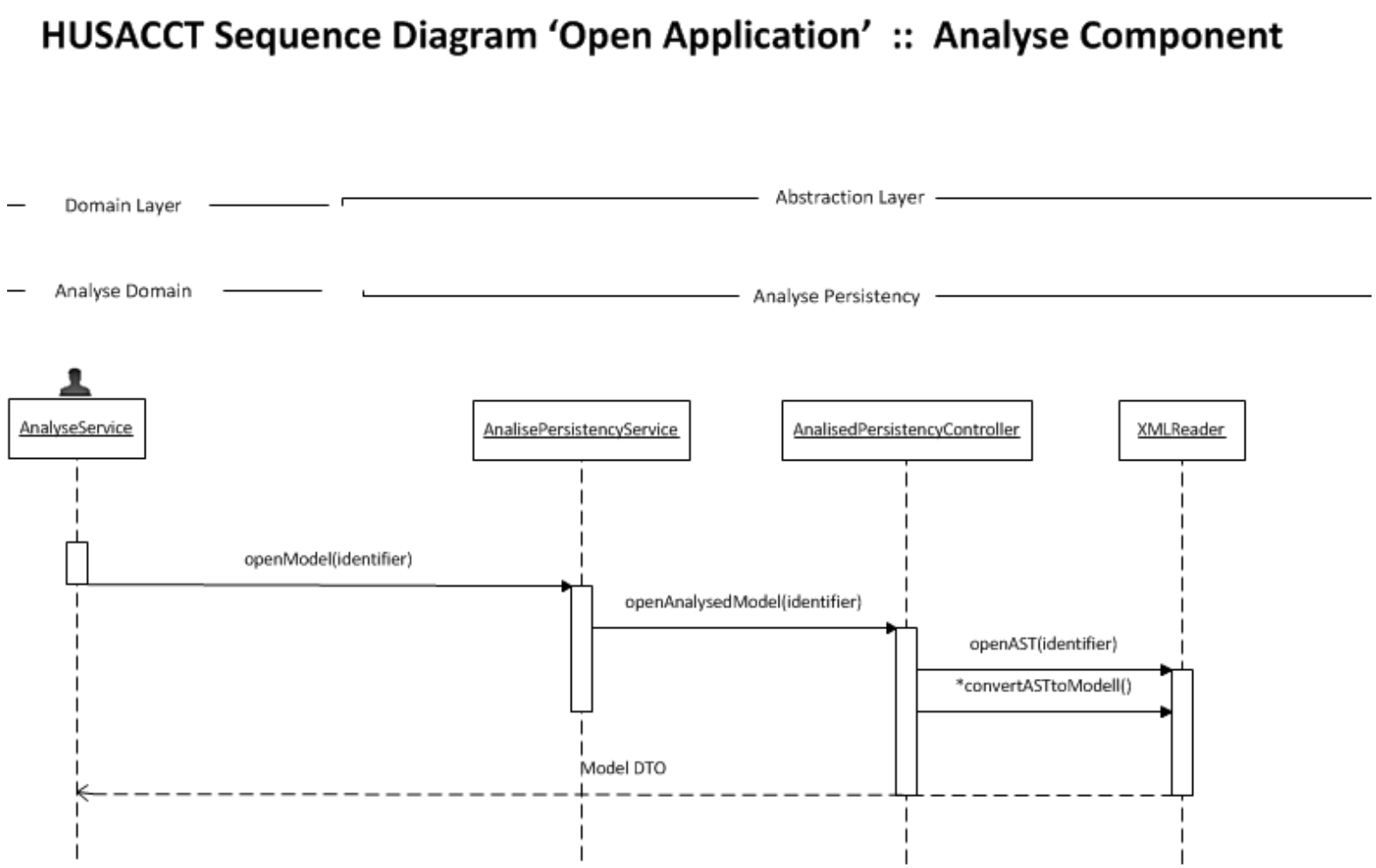

Figure 5.1. Abstract overview of the implementation of the use case Open Analysed Code