

# Design Document for Pipelines

Coen Stange, Edgar Kruze, Wen Li

2015/12/25

## Definitions and abbreviations

URS	User Requirements Specification document
DTO	Data Transfer Object
WPF	Windows Presentation Foundation
MVVM	Model View ViewModel
SOLID	Single responsibility, Open-closed, Liskov substitution, Interface segregation and Dependency inversion

## Bibliography

- [1] UML, *Unified Modeling Language (UML) version 2.5 specification document*, <http://www.omg.org/spec/UML/2.5/>

## Background and Context

This design document specifies the design decisions of the application "Pipelines in a flow network". It is recommended to read the URS first. The class diagrams and sequence diagrams are made according to the UML specifications see reference [1].

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Abstraction layers . . . . .	3
1.1.1	Common Layer . . . . .	4
1.1.2	Presentation Layer . . . . .	4
1.1.3	Business Layer . . . . .	4
1.1.4	Data Layer . . . . .	4
<b>2</b>	<b>Class diagram</b>	<b>5</b>
2.1	Common Layer . . . . .	5
2.2	Presentation Layer . . . . .	5
2.3	Business Layer . . . . .	5
2.4	Data Access layer . . . . .	6
<b>3</b>	<b>Sequence diagrams</b>	<b>12</b>
3.1	Business . . . . .	12
3.2	Overall . . . . .	12
3.3	Presentation layer . . . . .	12

# 1 | Introduction

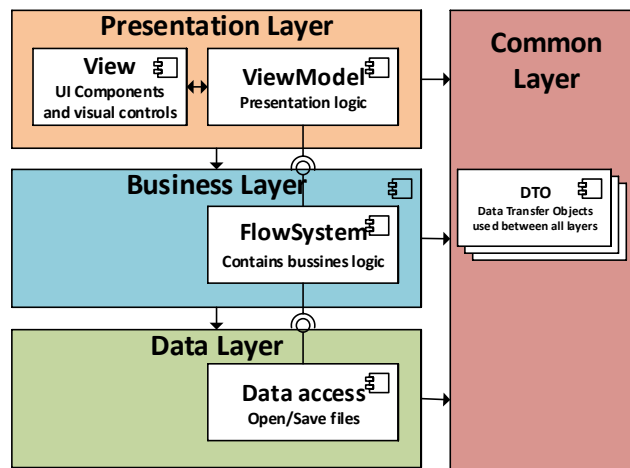


Figure 1.1: Overview of the system.

## 1.1 Abstraction layers

The system is divided in layers to make the software more modular. For example the "Data Layer" could be replaced to make it work with a Database, for now the application only works with the file system. The components of figure 1.1 know about each other through Dependency Injection whereby the components don't really know about each other to prevent the code from being glued together. The code is more testable since it is modular.

### 1.1.1 Common Layer

The Common Layer is referenced by all the other layers. It contains DTO's which are objects transferred between all the layers. The objects only contain data and don't have any behavior, so the objects are really stupid and have no logic.

### 1.1.2 Presentation Layer

The presentation layer handles the UI of the application, this layer doesn't have any business logic. WPF is used for handling the graphics since it allows an MVVM pattern to be implemented see figure 1.2. The Presentation Layer only contains the View and the ViewModel, the model is in the Bussines Layer. The MVVM pattern is used to change properties of selected components. For the graphics the Canvas element of WPF is used. It allows WPF to handle all the drawing. And the Canvas element allows events to be triggered when it has been pressed on.

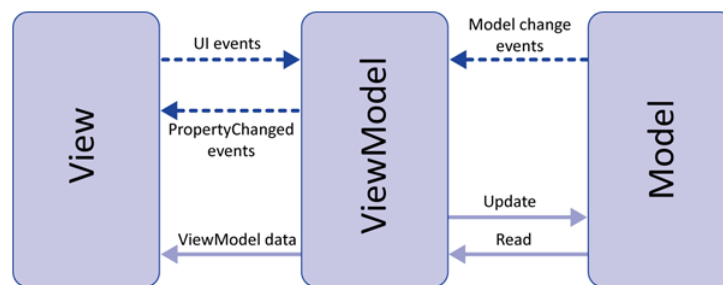


Figure 1.2: Model View ViewModel

### 1.1.3 Business Layer

The business layer contains the business logic of the program. Business logic include calculating the flow through the network.

### 1.1.4 Data Layer

The data layer contains the logic to read and write the flow network to a file. Since the Common layer can't be completely serialized the data has to be transformed. The problem lays with the relations between the components, since the interfaces can't be serialized. The solution in this application is to add an Unique identifier to each component and use those to specify relations in a file.

## 2 | Class diagram

### 2.1 Common Layer

The class diagram shown in figure 2.1 gives an overview of all the classes (but doesn't show all the relations!). This overview is to clarify that the *FlowNetworkEntity* have a "has-a" relationship with the components (*MergerEntity*, *PumpEntity*...).

In figure 2.2 the class diagram gives a more detailed view of the components and in more detail how the pipes are connected. Very noticeable are the interfaces *IFlowOutput* and *IFlowInput*. Since a component can have more than 1 input/output an array is used, whereby the pipe also needs to understand to which output it is connected hence the pipe also has the index pipe.

### 2.2 Presentation Layer

The presentation layer as shown in figure 2.3 has left out a lot of logic, e.g. click events and such since it is redundant to the reader. View-models are used to edit the properties of selected components. The *ViewModelBase* is an abstract class for all the view-models, using data-binding the data is shown on the screen. If the end-user edits a property an event will be triggered back. This isn't possible with the graphics since it works best with text and not with graphics manipulation. When data is changed it will be triggered to the business layer.

### 2.3 Business Layer

The class diagram of the Business Layer can be seen in figure 2.4 contains all the business logic. This layer has no classes derived from the components from the common layer (e.g. there is no *PumpModel*). That is since the business layer manipulates

the objects from the common layer. Like when the flow network needs to be updated it will use the implementation of *IFlowCalculator*.

## 2.4 Data Access layer

Figure 2.5 shows the class diagram of the Data Access layer, the Data Access Layer is small and it could be considered to move this to the business layer. When you open a file you get a dumb model back (objects from common) because otherwise the system is failing to follow Separation of Concerns in SOLID principals. The Data Access layer also has logic to transform data, not shown in the class diagram.

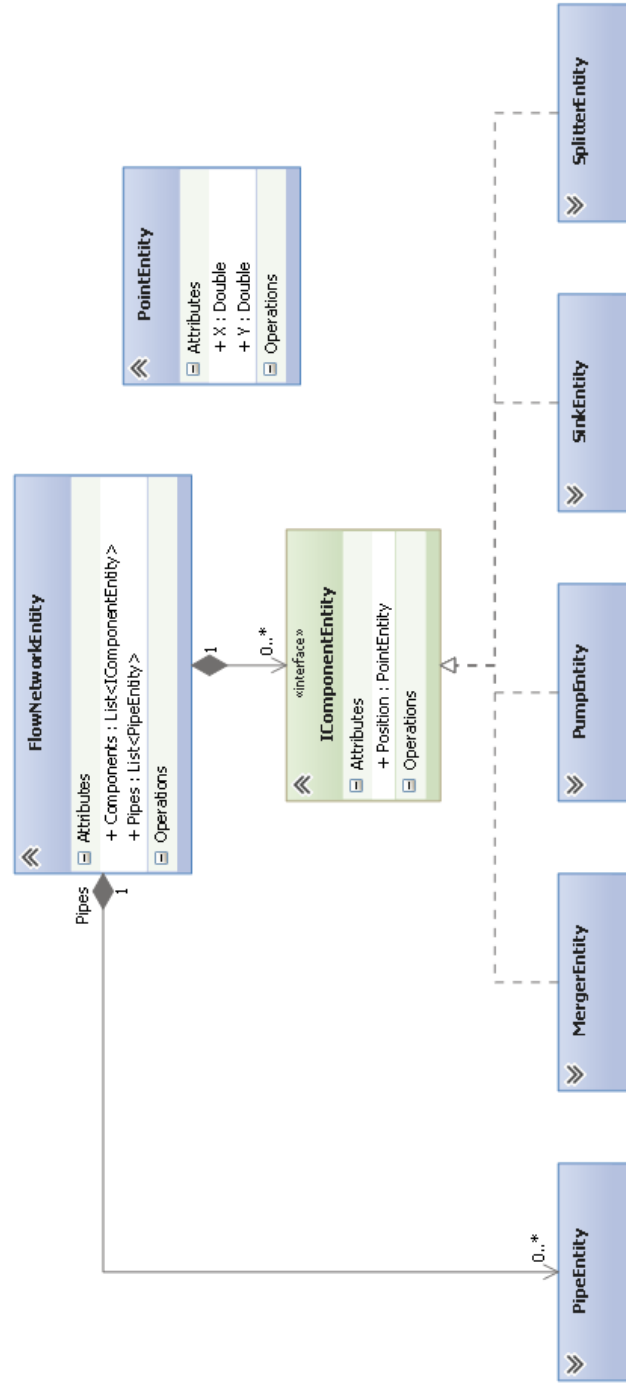


Figure 2.1: Class Diagram Common overview



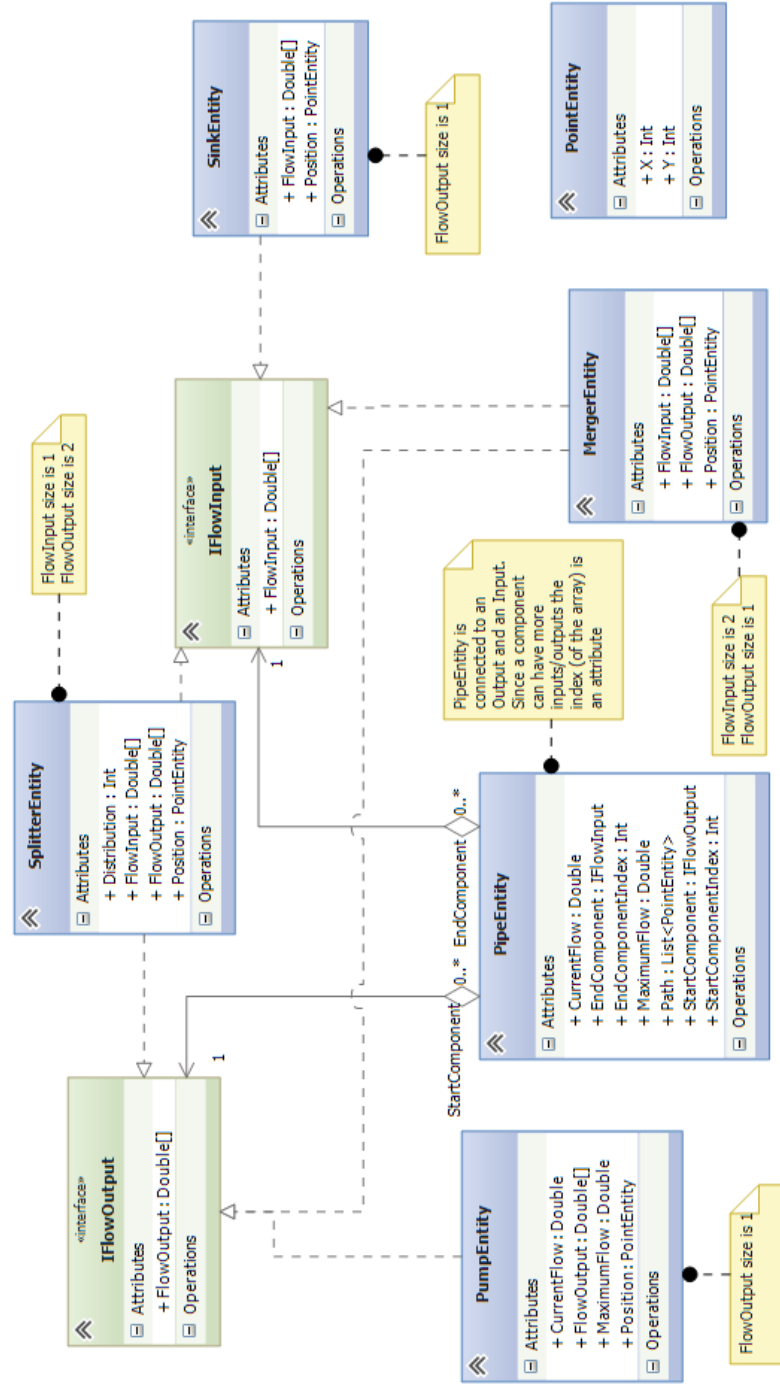


Figure 2.2: Class Diagram Common Components

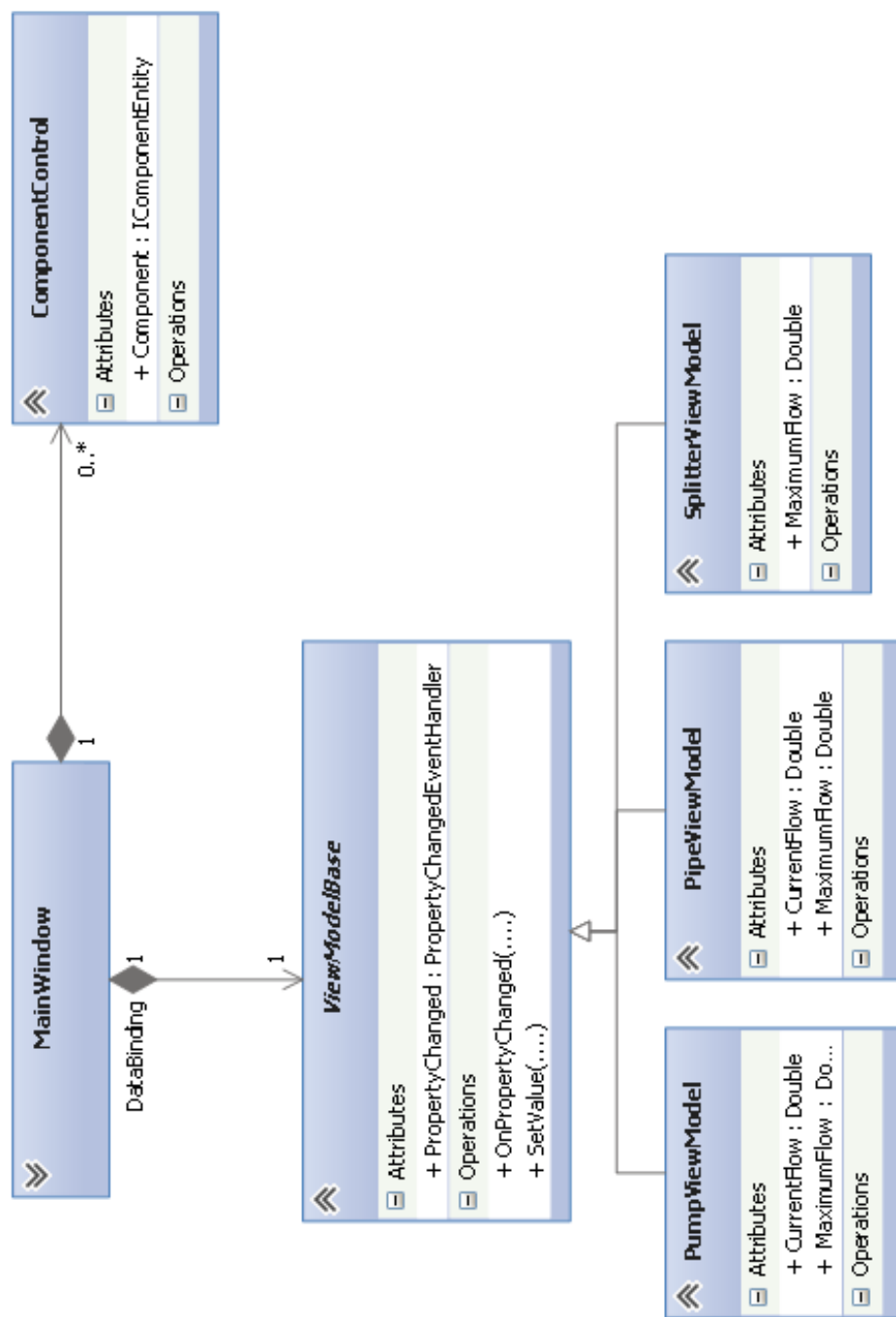


Figure 2.3: Class Diagram Presentation Layer

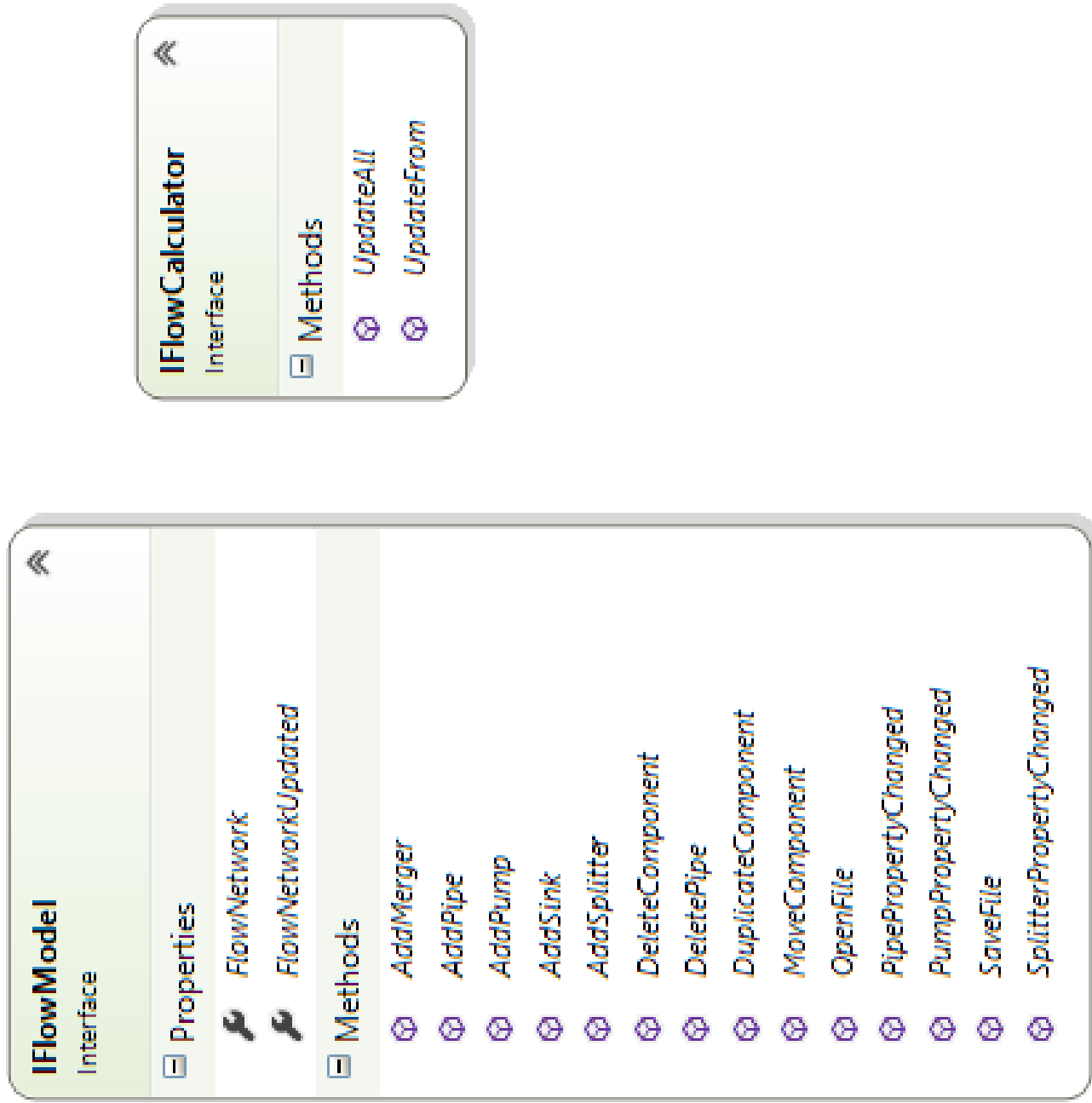


Figure 2.4: Class Diagram Business Layer

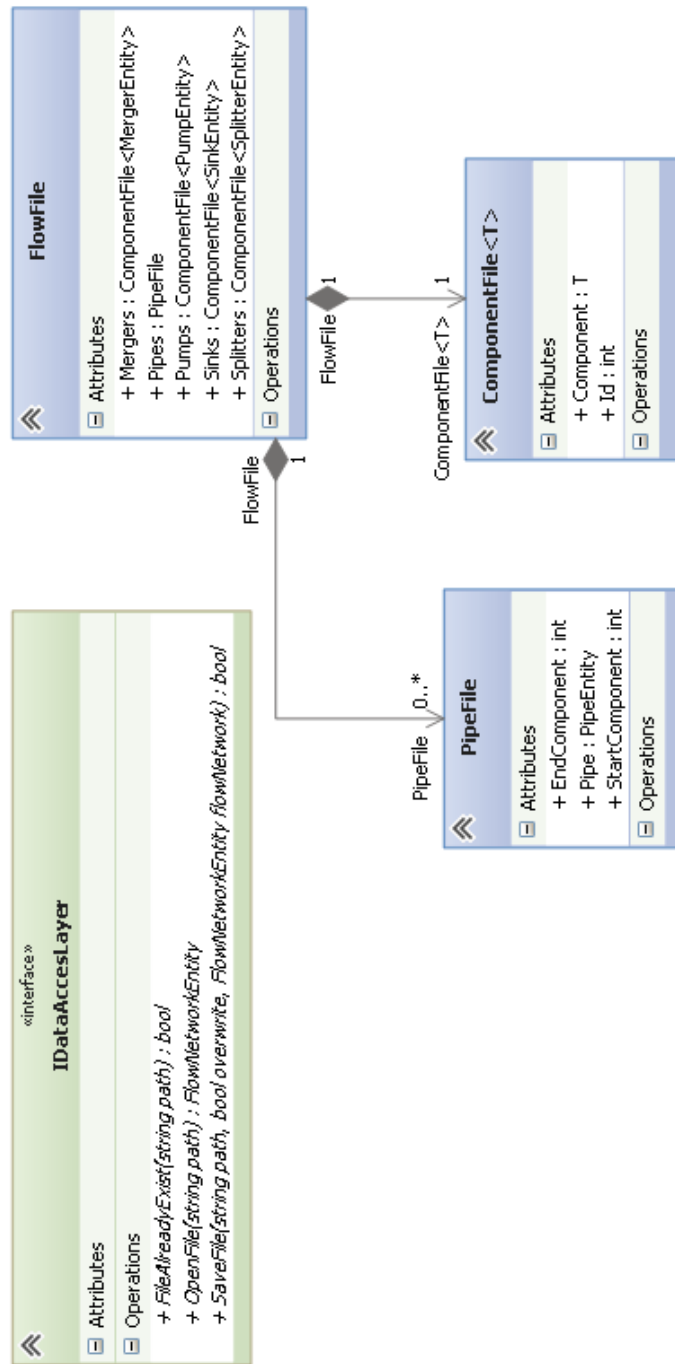


Figure 2.5: Class Diagram Data Access Layer

## 3 | Sequence diagrams

### 3.1 Business

The business layer isn't supposed to know about the view. As seen in figure 3.1 the *MainWindow* is not visible, so the message comes from 'somewhere'. Then it will do some business logic like checking if a component would not be overlapping another component. At the end when the flow network has changed the changes will be sent back using callbacks to the presentation layer.

### 3.2 Overall

In figure 3.2 a sequence diagram is visible which goes through all the layers of the application. Very noticeable is that *IFlowModel* and *IDataAccessLayer* both have the *OpenFile* operation. Usually the presentation layer isn't supposed to know about the data layer because the data access layer usually contains CRUD operations.

### 3.3 Presentation layer

In figure 3.3 many things are triggered by the user. Those are triggered by events from WPF when a component is selected. The *flowModel* contains an event which has to be triggered when the network has changed. That event is passed on to the *IFlowCalculator* to be called when it is done.

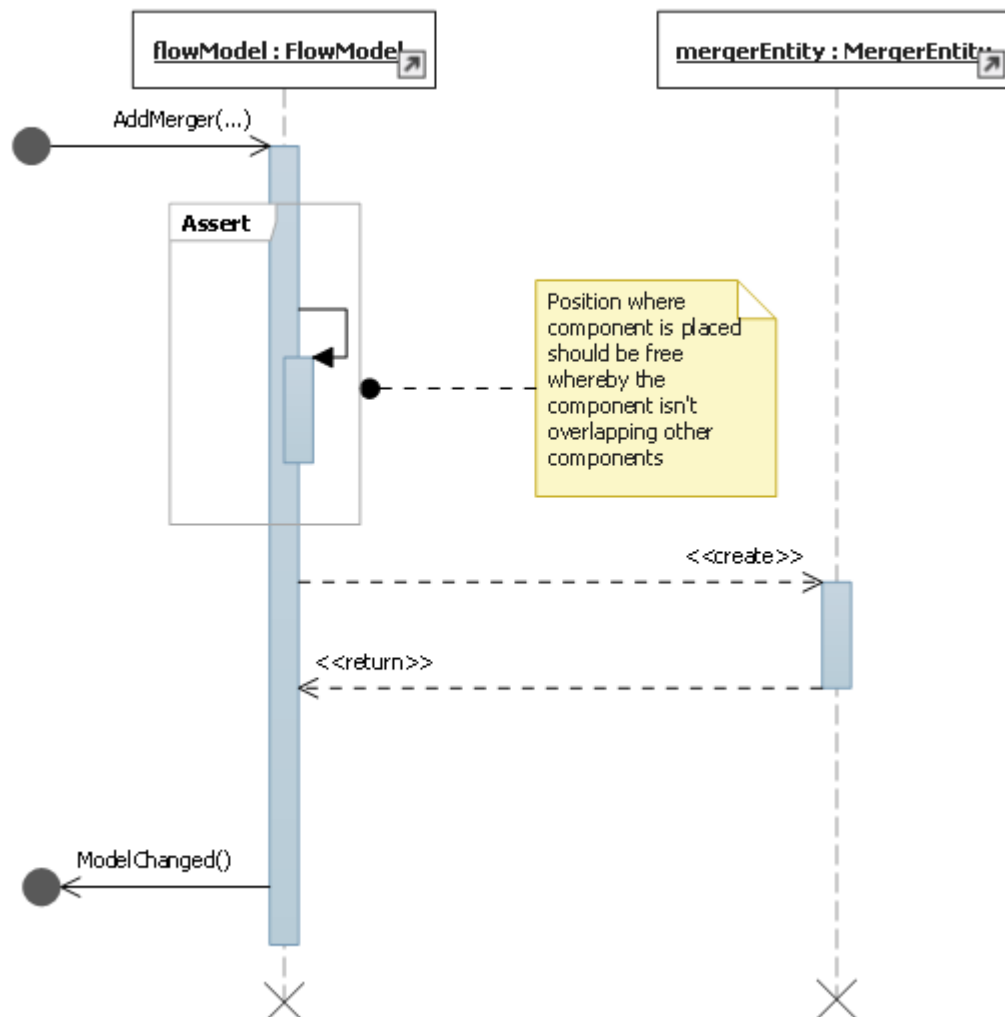


Figure 3.1: Business add new component

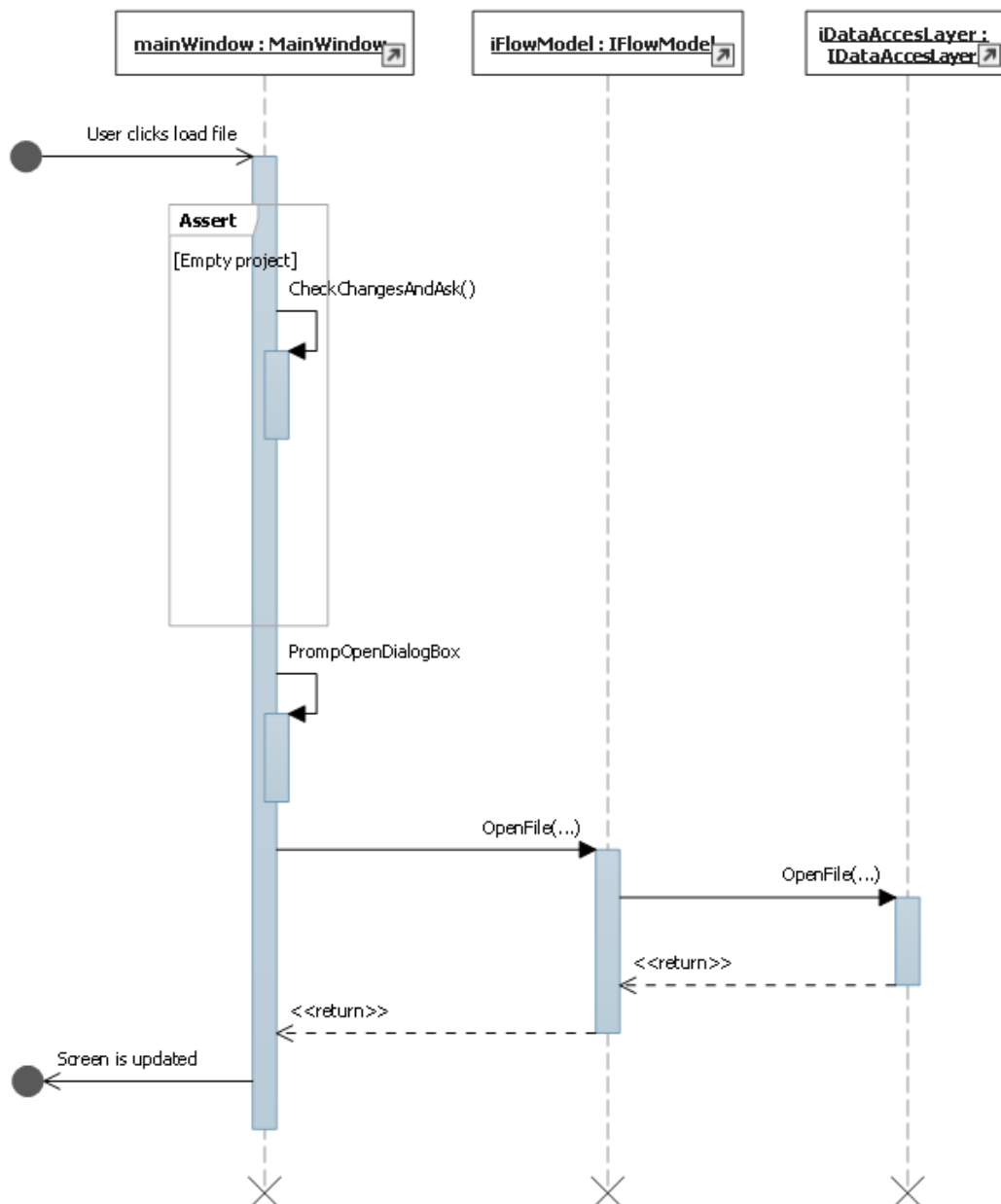


Figure 3.2: Overall load file

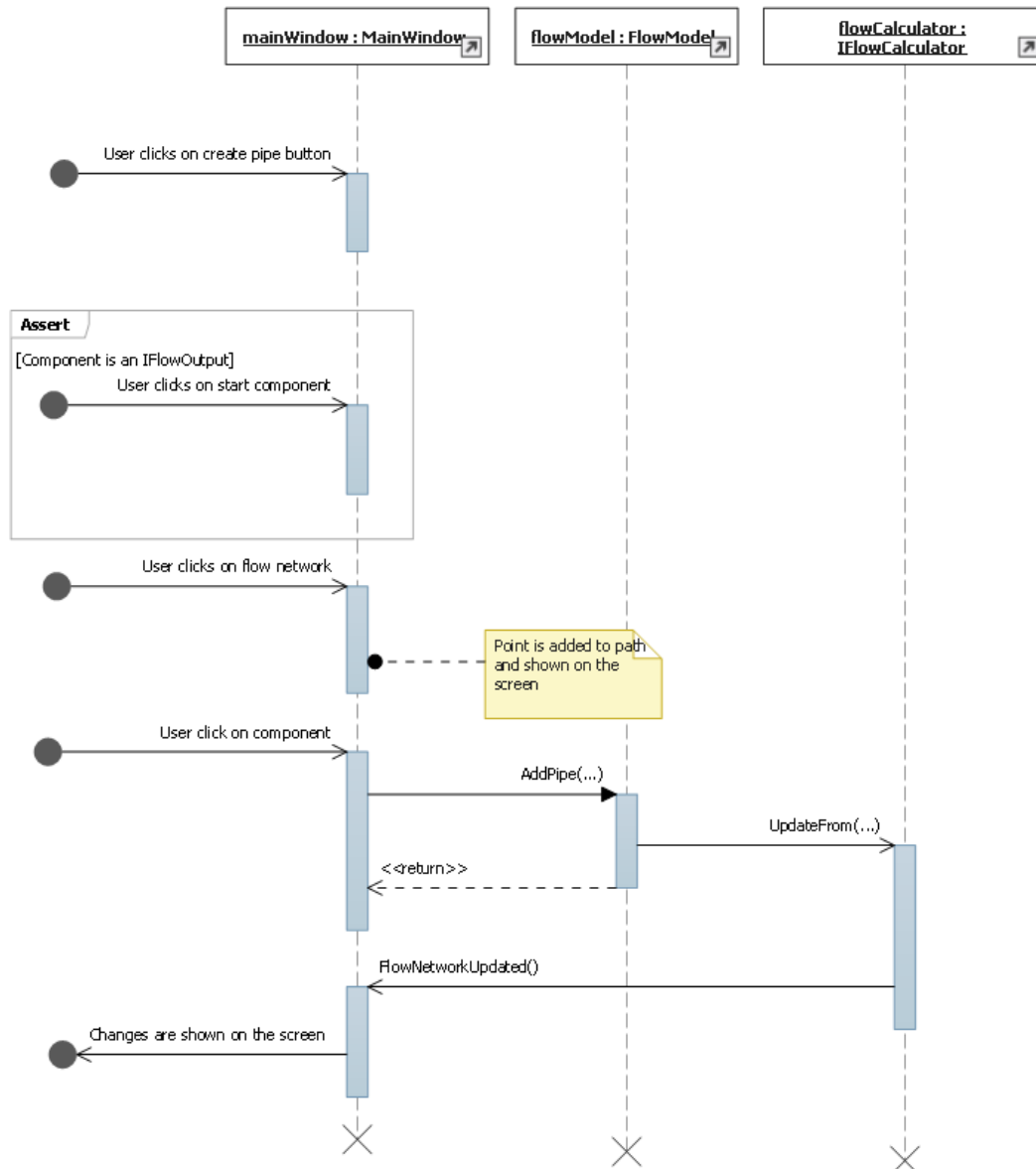


Figure 3.3: Presentation create pipeline