Fontys University of Applied Sciences

Design Document for Pipelines

Coen Stange, Edgar Kruze, Wen Li

2015/12/09

Definitions and abbreviations

URS	User Requirements Specification document
DTO	Data Transfer Object
WPF	Windows Presentation Foundation
MVVM	Model View ViewModel
MVC	Model View Controller
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 specifcations see reference [1].

Contents

1 Introduction

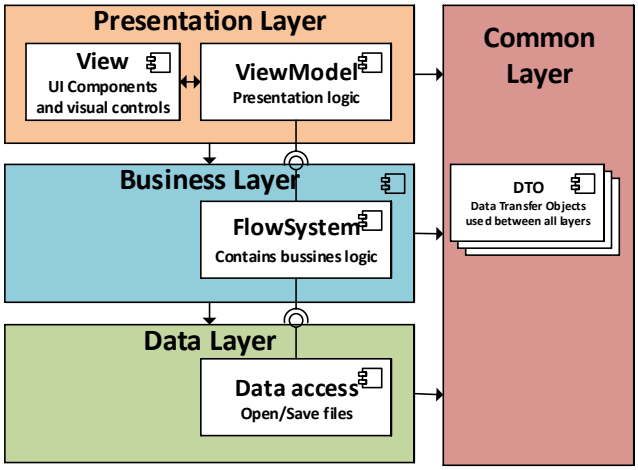


Figure 1.1: Overview of the system.

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 it is the best option to use the build-in graphics library and use that in a MVC pattern since it doesn't support MVVM.

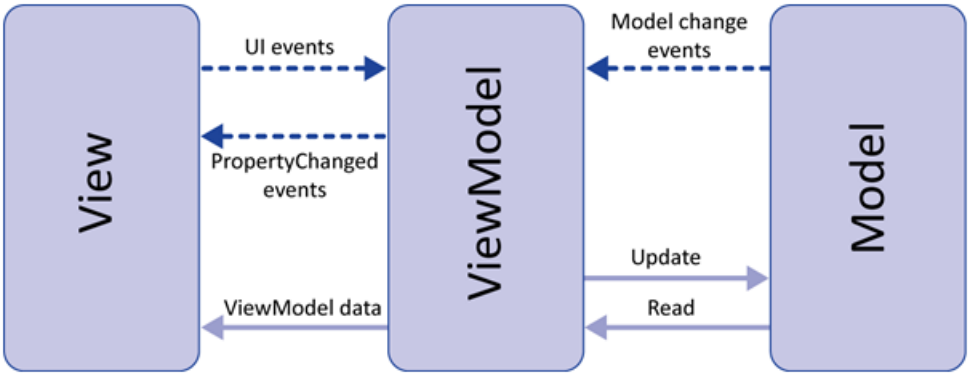


Figure 1.2: Model View ViewModel

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.

2 | Class diagram

2.1 Common Layer

The class diagram shown in figure 2.2 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 2.3 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 2.4 has left out a lot of logic, e.g. click events ands 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 2.5 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 2.1 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.

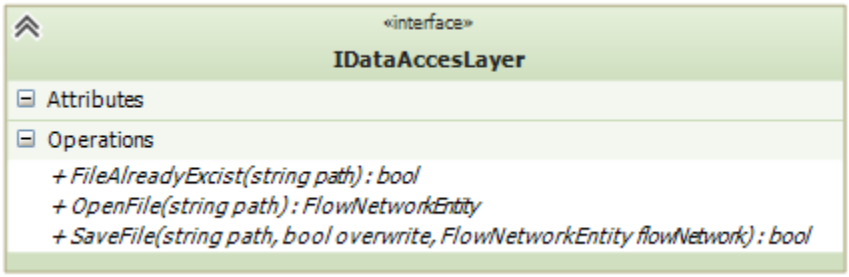


Figure 2.1: Class Diagram Data Access Layer

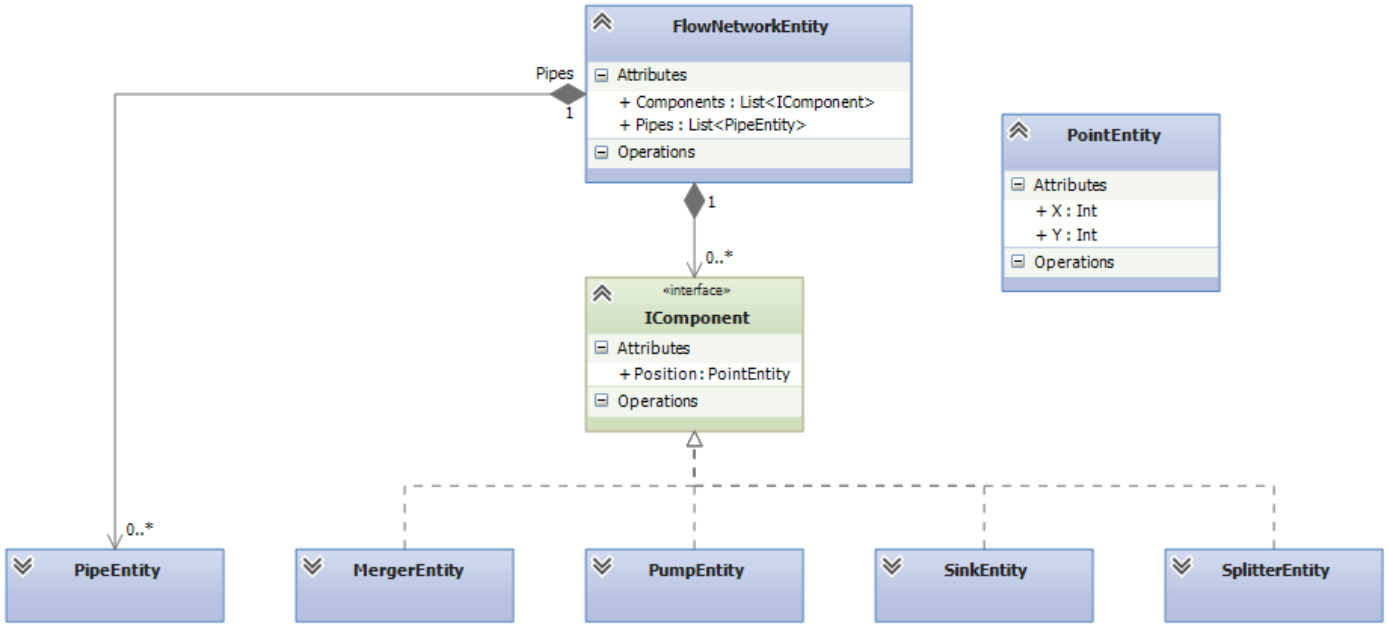


Figure 2.2: Class Diagram Common overview

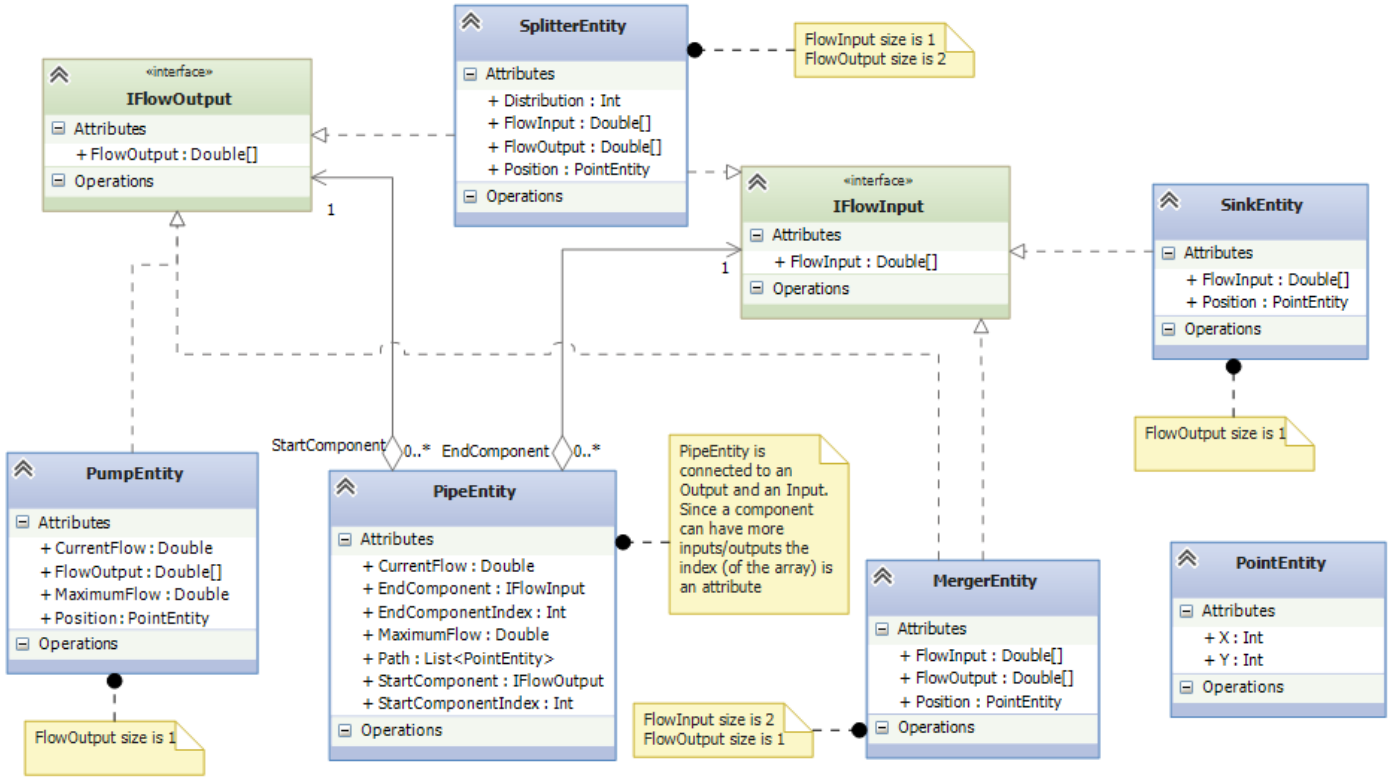


Figure 2.3: Class Diagram Common Components

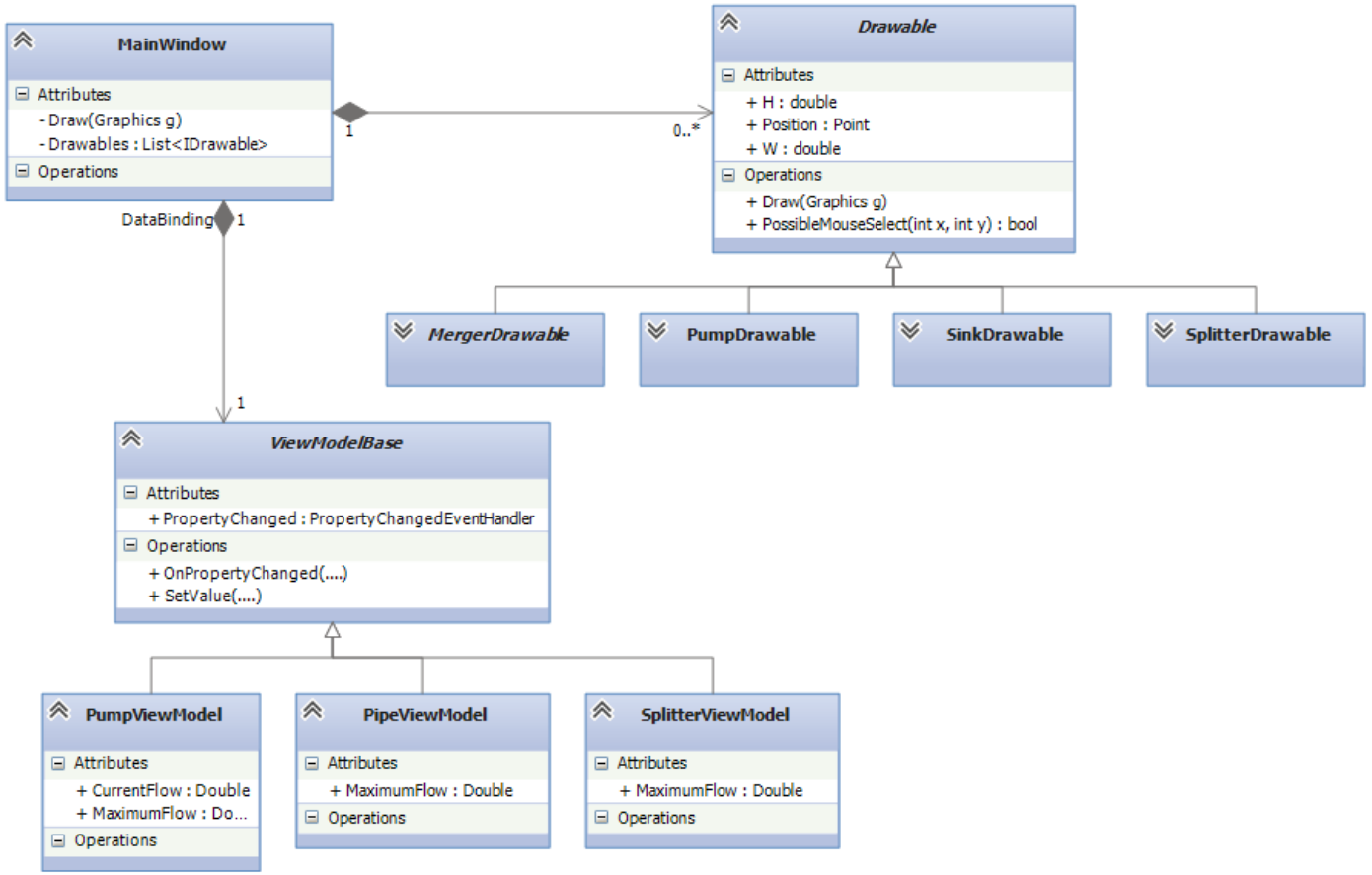


Figure 2.4: Class Diagram Presentation Layer

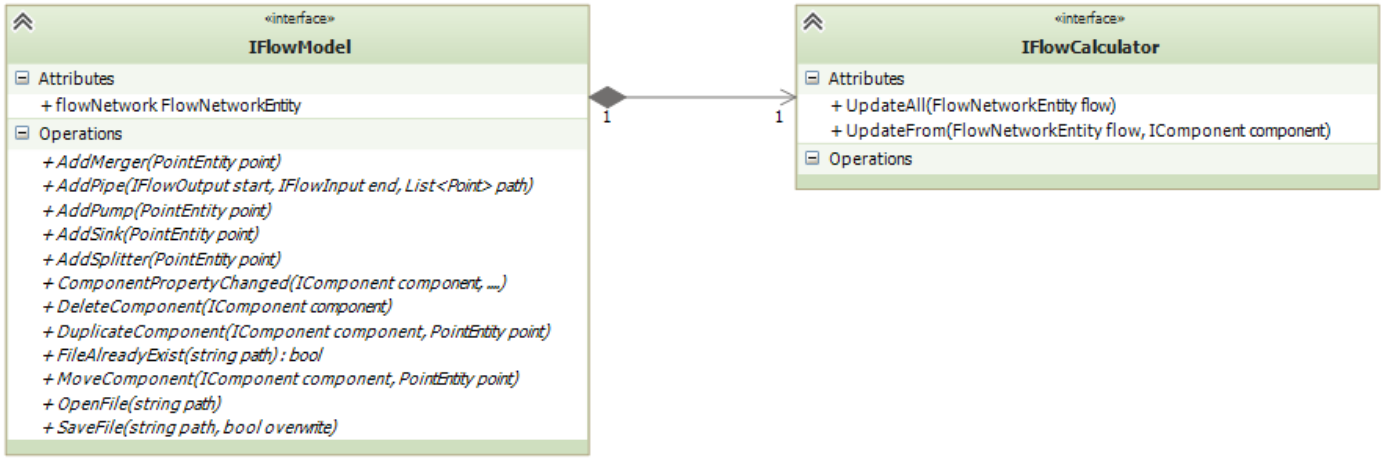


Figure 2.5: Class Diagram Business Layer

3 | Sequence diagrams

3.1 Business

The businness 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 bushiness 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 send 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 *IDataAccesLayer* both have the *OpenFile* operation. Usually the presentation layer isn't suppose to know about the data layer because the data access layer usually contains CRUD operations.

3.3 Presentation layer

The sequence diagram in figure 3.3 only contains the MVC pattern since no *View-Model* is used. Many steps are required to get the selected component on the screen since the standard graphics are used.

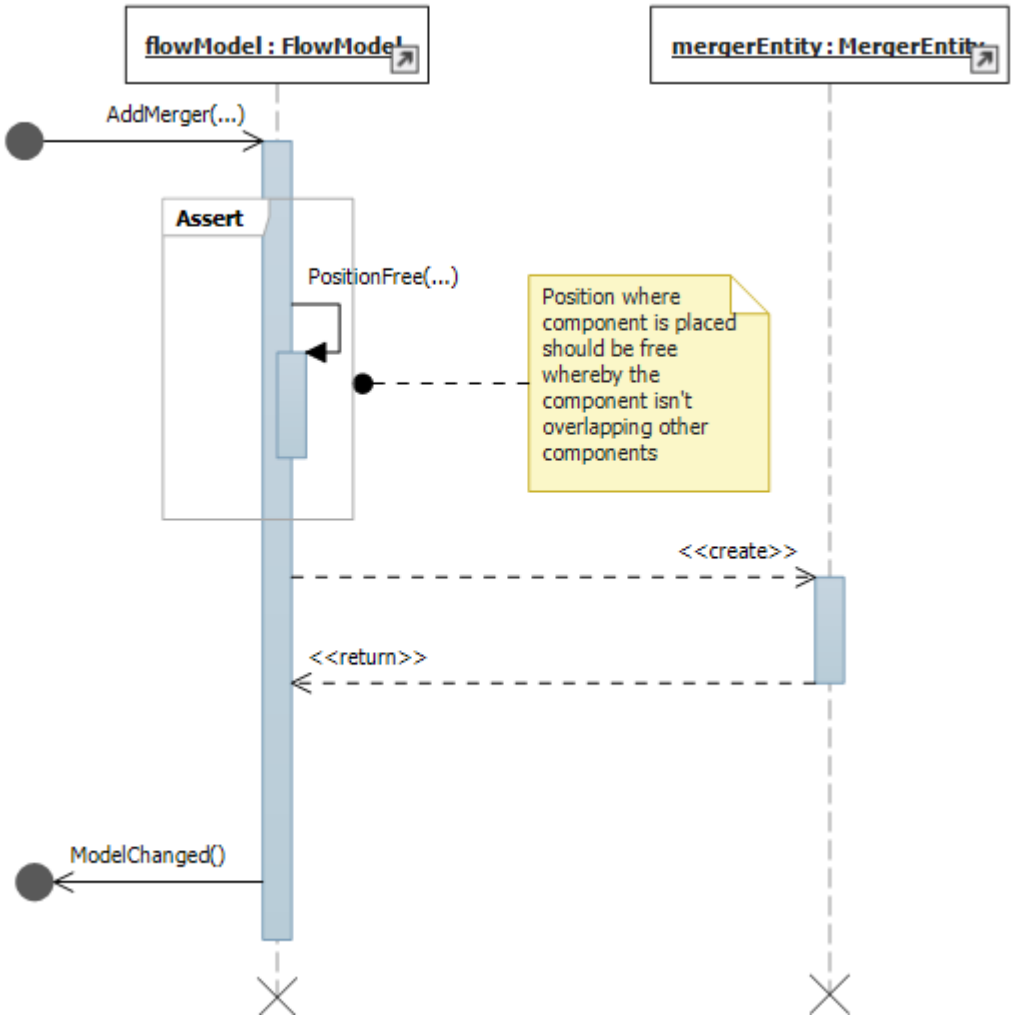


Figure 3.1: Business add new component

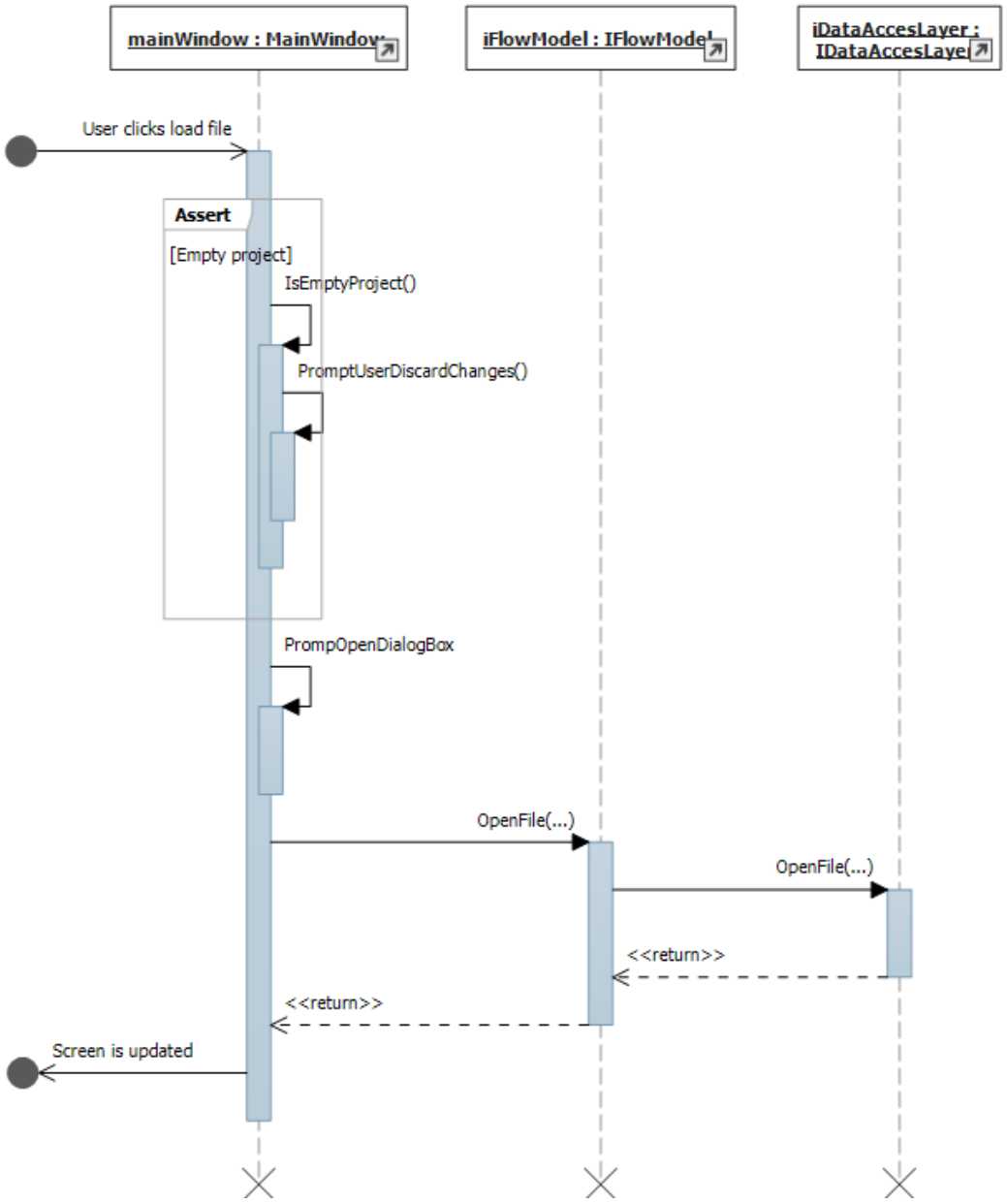


Figure 3.2: Overall load file

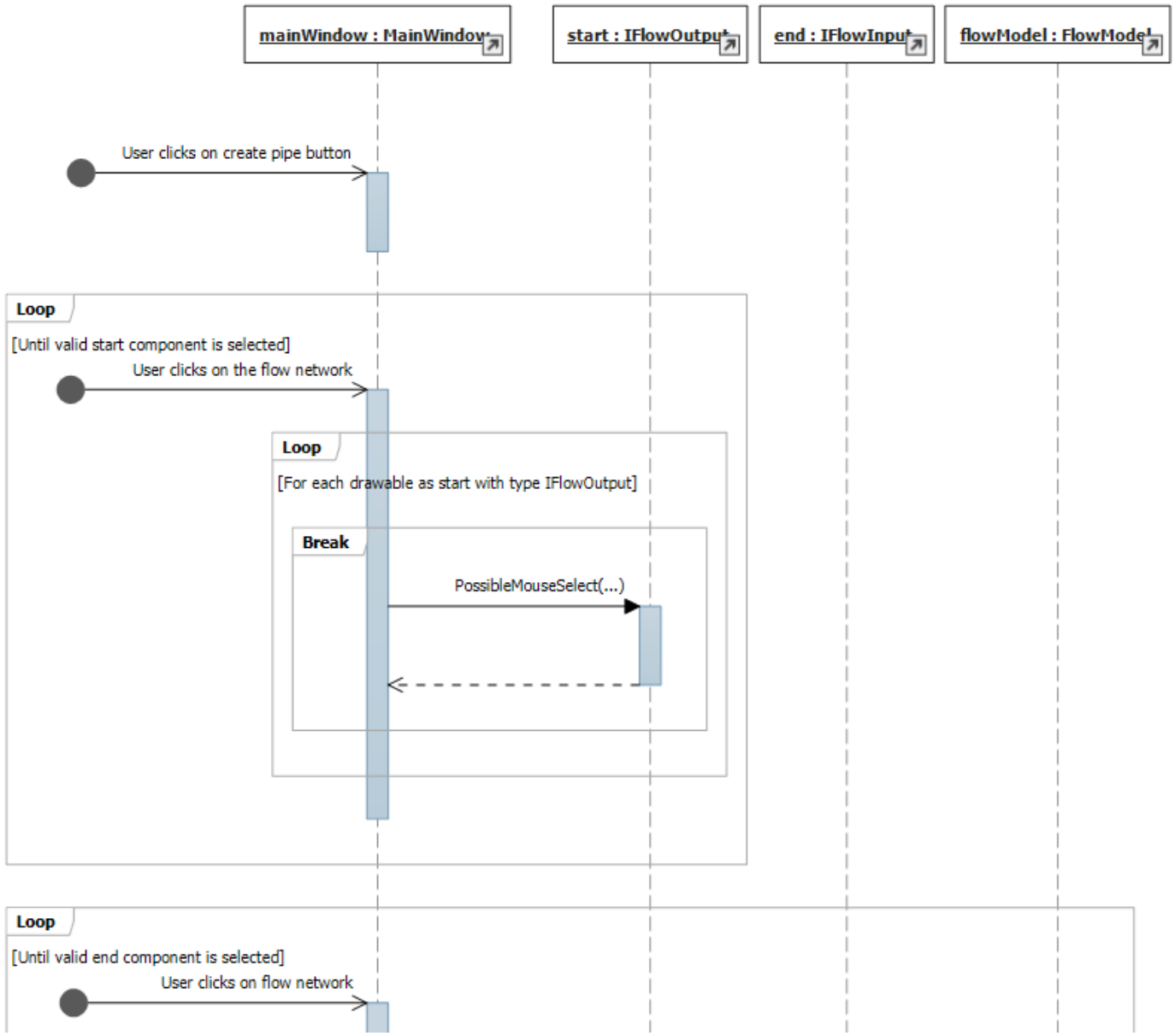
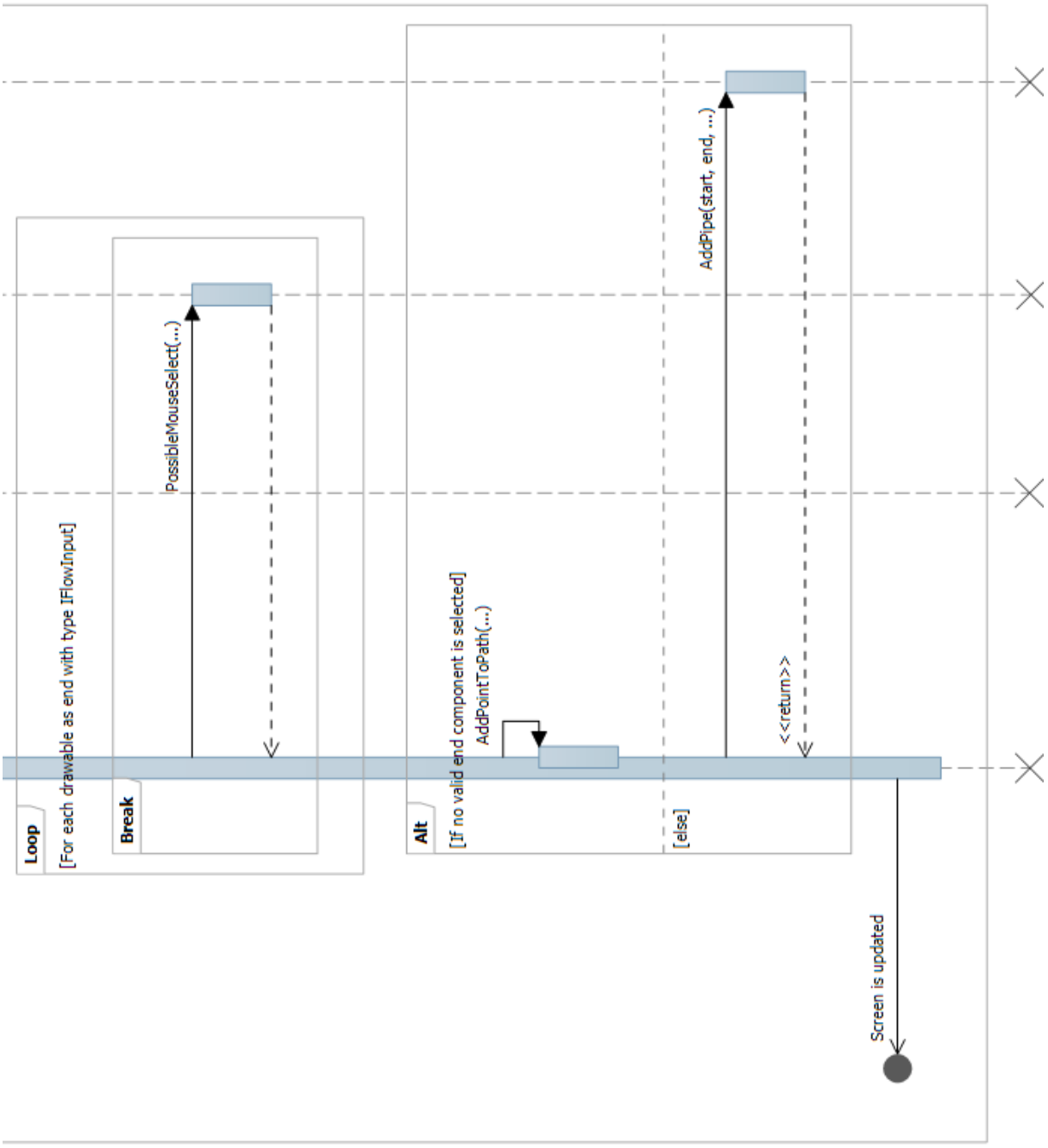


Figure 3.3: Presentation create pipeline


Loop
[For each drawable as end with type IFlowInput]
Break
PossibleMouseSelect(...)
Alt
[If no valid end component is selected]
AddPointToPath(...)
[else]
AddPipe(start, end, ...)
<<return>>
Screen is updated