Knowledge Based Engineering - Assignment 3

Automatic pipe routing for Aker Solutions

Spring 2022

Hallvard Bjørgen
Johanne Glende
Sigve Sjøvold

Table of Contents

Product description

Comprehension story

The automatic pipe routing system is created to automatically find the shortest and best path from A to B, intersecting all necessary equipment in the environment.

Investing in the system will involve:
- Measuring and providing the input of:
1. The environment space.
2. Point A, where path will begin.
3. Point B, where path will end.
4. Coordinates of all equipment the piping must intersect.

In return one will obtain a supposal of where the piping might go to:
- Start in A and end in B.
- Intersect neccessary equipment.

The solution will be sent as a DFA-file (runnable in Siemens NX) to the email provided by the user.

Selected parameters

The user interface is “under-developed”. Time has gone to other parts of the development for this assignment.

Selected parameters are thus all the variables that can be changes and then produce different visible solutions. We’ve not gotten to the point where the application produces non-visible information, or does any kind of input-control. The application could be extended to check what the pressure would be in the pipe with given inner diameter and/or what material needs to be used, how much material is needed, what the heat-loss will be etc.

The selected parameters are then all physically visible variables for environment, equipments and pipe, as well as the email one wants their solution sent to (the file is also put in a folder “/inputOutput/products”).

Application architecture

Model-view-controller

We have mostly stuck to a model-view-controller architectural pattern for code simplicity. We do, however, not have a view (the closest is our runnable “UserInterface.py” in the top level folder).

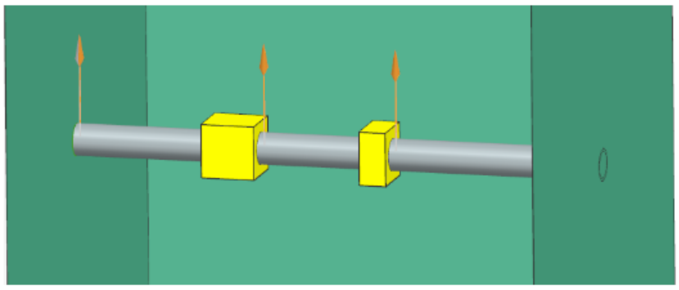


Figure 1: simple straight example

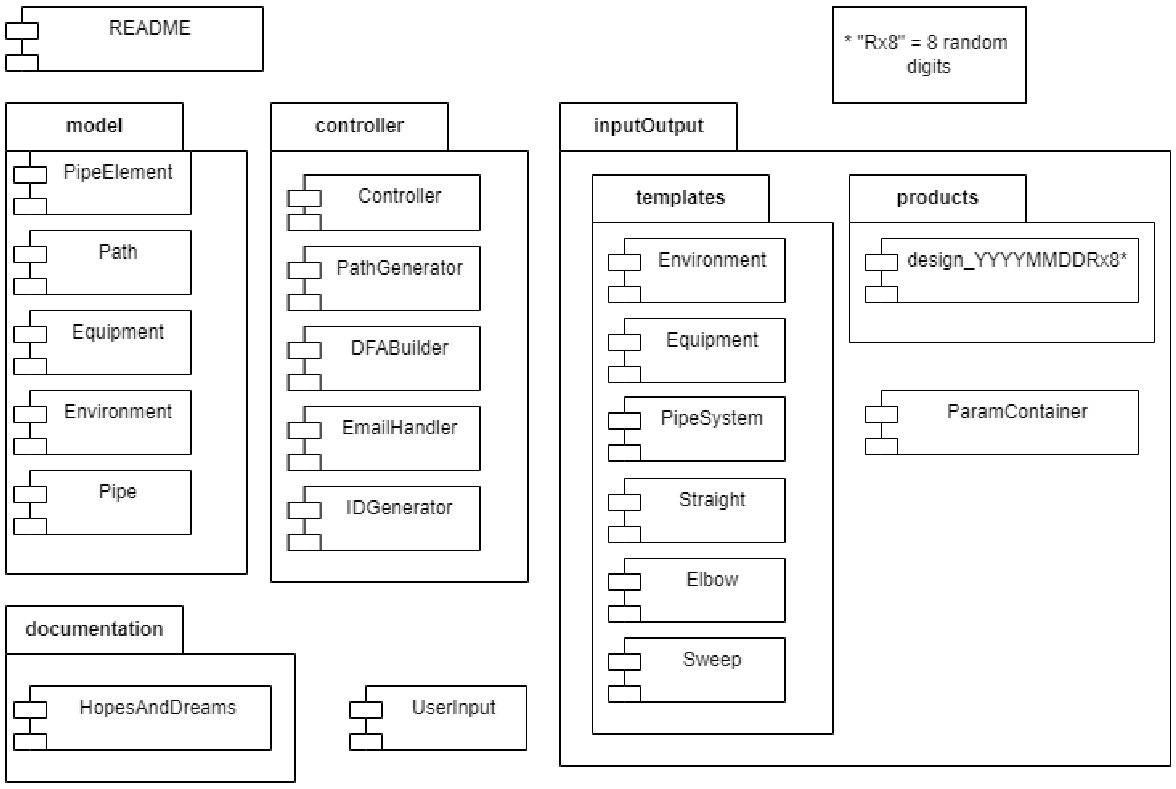


Figure 2 : package diagram

Model

The model package contains all logic, and consists of Environment, Equipment, Path, Pipe, and PipeElement. In fig. 1: “simple straight example”, the environment is green, equipment is yellow, path is invisible (but is the middle of where the piping will go) and is, in fig. 1, split so there’s three paths in fig. 1. Pipe is the cross section of the pipe (and the radius of elbow elements, not shown in fig. 1), and the pipe element is the three straight pipings between equipments portrayed in grey.

View

Our application requires only the UserInterface.py to be fed input, and instead of a separate Main file running the application, the files are merged to simplify the process of running new input. The UserInterface file is therefore put on top level, and the view package nonexistent.

Controller

The controller package consists of classes that are used to integrate and pass information. For our application, a path is the pipe between an inlet and an outlet. In fig. 1, three paths are shown.

- **Controller**

Is related to userinterface, passing information to other controllers.

- **DFABuilder**

Interacts with the dfa templates to create the product dfa file.

- **EmailHandler**

Sends the product to the email defined in UserInterface.

- **IDGenerator**

Generates random IDs (and is a controller because of it’s aspirations to accumulate information from other classes and make unique ID from this).

- **PathGenerator**

Handles the complete path from input of the environment to output of the environment, splitting into different paths and having them generate their own new paths out of path elements, which are either straights or elbows.

inputOutput

This package consists of the template files for the dfa file creation, and our ParameterContainer. In addition, the products are also stored in a folder here “/products”, which is created should it not be existent prior to running the application.

Application interaction

To illustrate and further supplement the interaction discussed above:

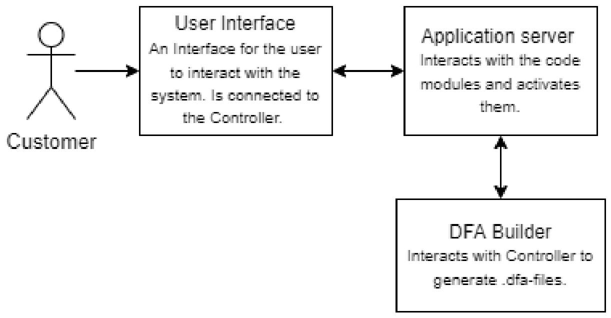
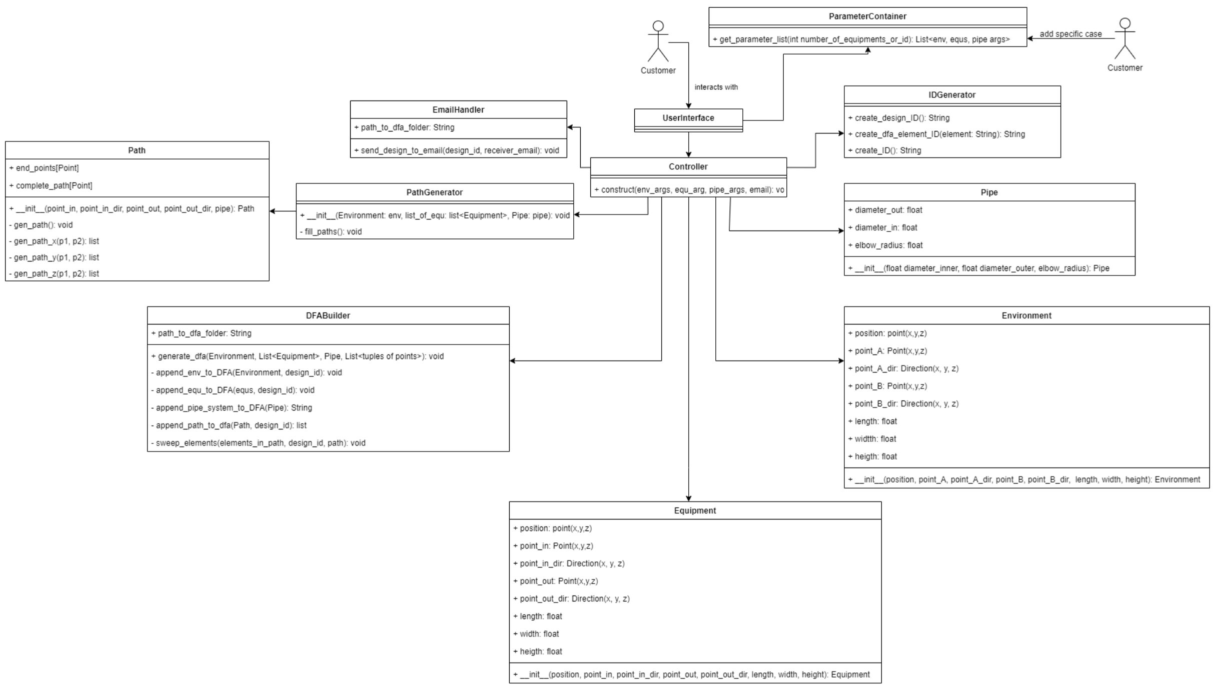

Customer
User Interface
An Interface for the user to interact with the system. Is connected to the Controller.
Application server
Interacts with the code modules and activates them.
DFA Builder
Interacts with Controller to generate .dfa-files.

Figure 3: Architecture diagram

More UML diagrams

Check out all the diagrams (in higher resolution) here!
https://drive.google.com/file/d/1aewpo-coM41aiBa-a9W4_Jk235NSWVIS/view?usp=sharing

Design class diagram



ParameterContainer
+ get_parameter_list(int number_of_equipments_or_id): List<env, equs, pipe args>
add specific case
Customer
Customer
interacts with
UserInterface
IDGenerator
+ create_design_ID(): String
+ create_dfa_element_ID(element: String): String
+ create_ID(): String
EmailHandler
+ path_to_dfa_folder: String
+ send_design_to_email(design_id, receiver_email): void
Controller
+ construct(env_args, equ_arg, pipe_args, email): vo
Path
+ end_points[Point]
+ complete_path[Point]
+ __init__(point_in, point_in_dir, point_out, point_out_dir, pipe): Path
- gen_path(): void
- gen_path_x(p1, p2): list
- gen_path_y(p1, p2): list
- gen_path_z(p1, p2): list
PathGenerator
+ __init__(Environment: env, list_of_equ: list<Equipment>, Pipe: pipe): void
- fill_paths(): void
Pipe
+ diameter_out: float
+ diameter_in: float
+ elbow_radius: float
+ __init__(float diameter_inner, float diameter_outer, elbow_radius): Pipe
DFABuilder
+ path_to_dfa_folder: String
+ generate_dfa(Environment, List<Equipment>, Pipe, List<tuples of points>): void
- append_env_to_DFA(Environment, design_id): void
- append_equ_to_DFA(equs, design_id): void
- append_pipe_system_to_DFA(Pipe): String
- append_path_to_dfa(Path, design_id): list
- sweep_elements(elements_in_path, design_id, path): void
Environment
+ position: point(x,y,z)
+ point_A: Point(x,y,z)
+ point_A_dir: Direction(x, y, z)
+ point_B: Point(x,y,z)
+ point_B_dir: Direction(x, y, z)
+ length: float
+ widtth: float
+ heigth: float
+ __init__(position, point_A, point_A_dir, point_B, point_B_dir, length, width, height): Environment
Equipment
+ position: point(x,y,z)
+ point_in: Point(x,y,z)
+ point_in_dir: Direction(x, y, z)
+ point_out: Point(x,y,z)
+ point_out_dir: Direction(x, y, z)
+ length: float
+ width: float
+ heigth: float
+ __init__(position, point_in, point_in_dir, point_out, point_out_dir, length, width, height): Equipment

Figure 4: Design class diagram

Problem domain class diagram

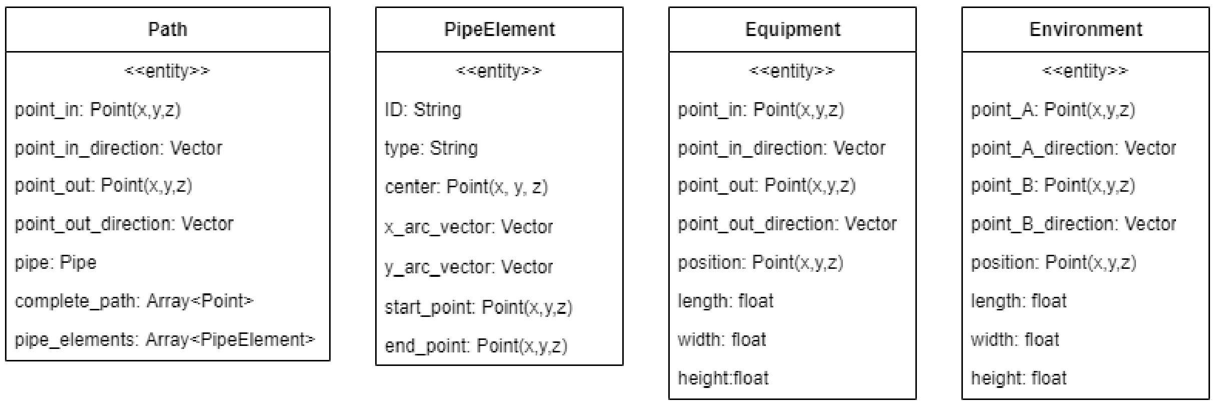


Figure 5: Problem domain class diagram

Sequence diagram: Customer provides number of equipments

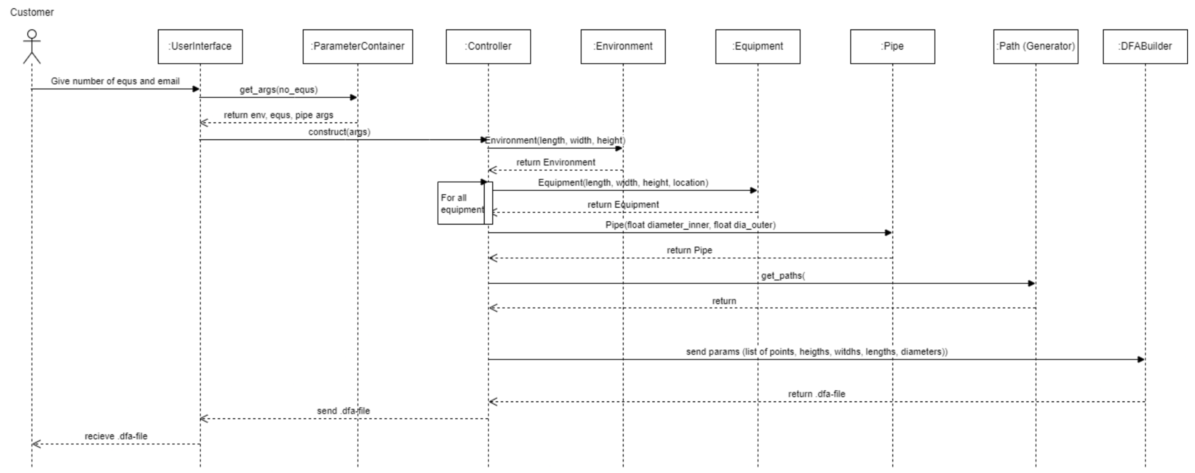


Figure 6: Sequence: customer gets random solution

Sequence diagram: Engineer creates new case

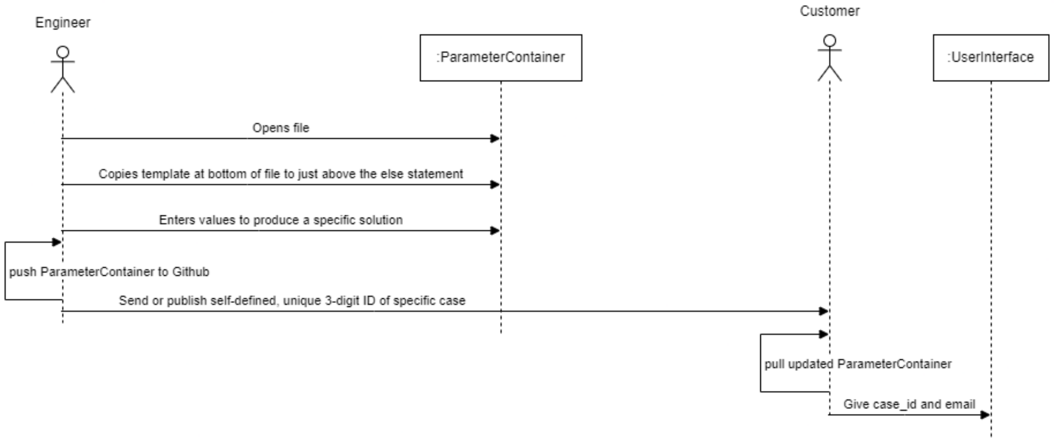

Engineer
:ParameterContainer
Customer
:UserInterface
Opens file
Copies template at bottom of file to just above the else statement
Enters values to produce a specific solution
push ParameterContainer to Github
Send or publish self-defined, unique 3-digit ID of specific case
pull updated ParameterContainer
Give case_id and email

Figure 7: Sequence: engineer creates new case

For figure 7, the rest of the sequence is the same as in figure 6.

Activity diagram

Figure 8 has swim lanes for User, Server, and DFA Builder with activity entities related to the main use cases of the application.

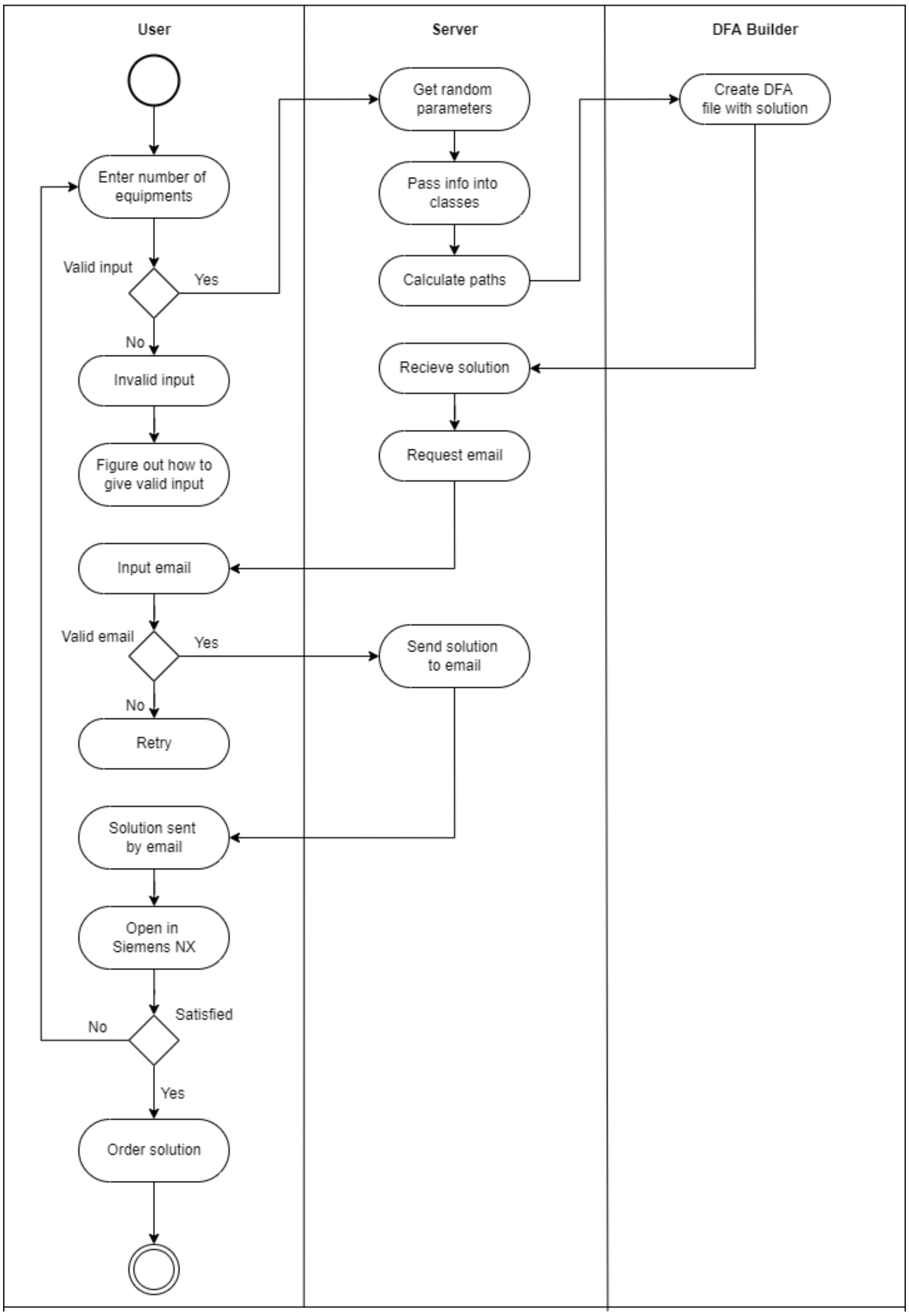


Figure 8: Activity diagram

Developed code

Controller modules

All controllers interactions are described in controller, under application architecture, which gives a sufficient understanding of their purpose and main functions.

Model modules

- **Environment**

Instantiates the environment used in the product, with the parameters specified in ParameterContainer.

- **Equipment**

Instantiates the different pieces of equipment.

- **Path**

Instantiates the path. Also contains functions for calculating if the path contains a single straight or two straights with an elbow. Also instantiates the elbows with the correct orientation.

- **Pipe**

Instantiates the pipe profile

- **PipeElement**

Instantiates the pipe element either as an elbow or a straight

InputOutput modules

- **ParameterContainer**

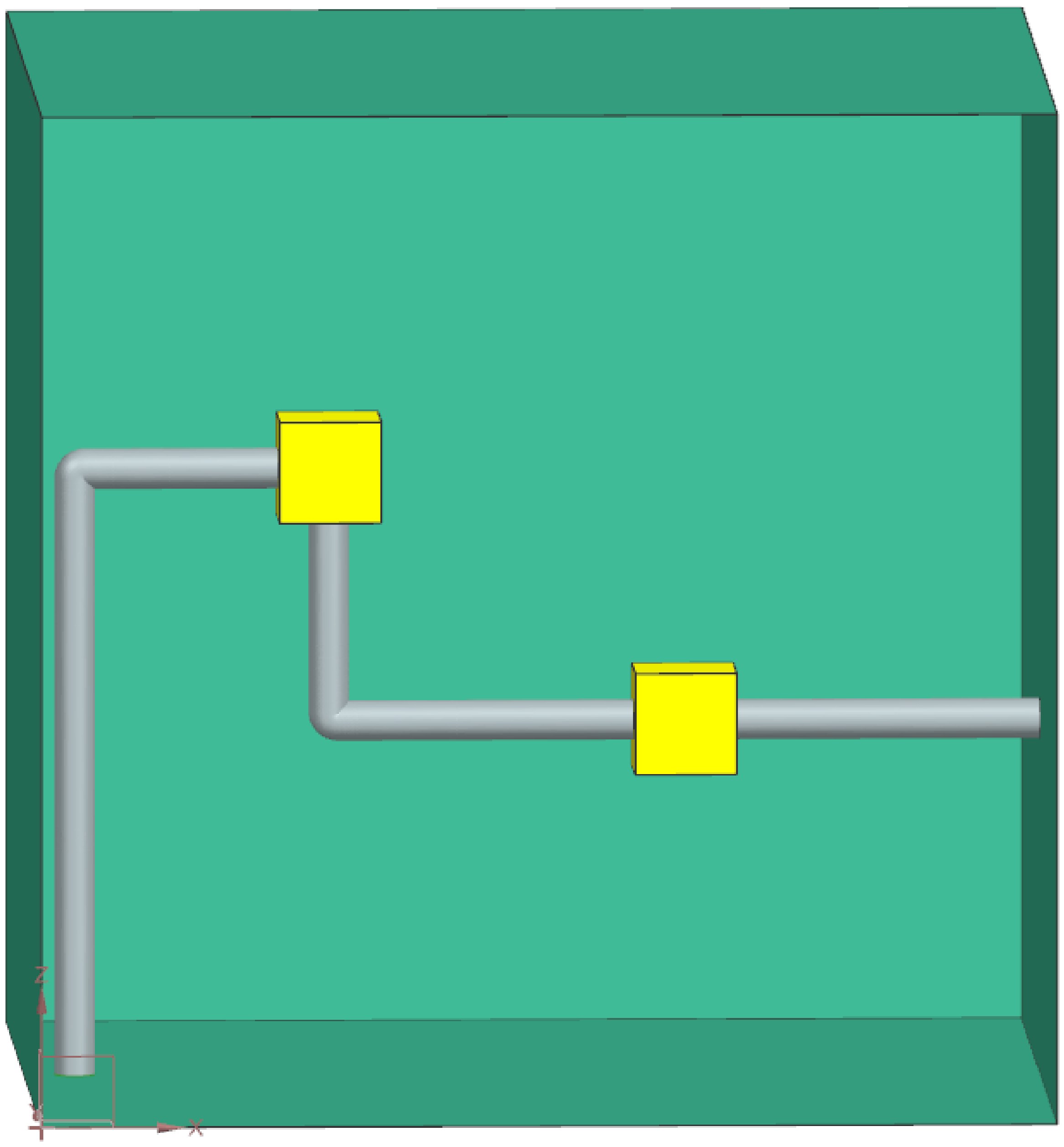
is the storage of predefined parameters providing the application with functional solutions, allowing the user to define how many equipments should be in the solution, and get one of the solutions having this number of equipments.

There should be a way to use the ID-defined solutions when only specifying number of equs as well. The parameter-combinations defined by engineers could have been randomly generated instead, but this would need more work to be created. If the solution is how it is now, the solutions the engineers create should rather be added to a knowledge base where one could easily extract all solutions with a given number of equipments, specific environment characteristics (like "my environment is less than ..."), or the one solution with a specific ID.

Results

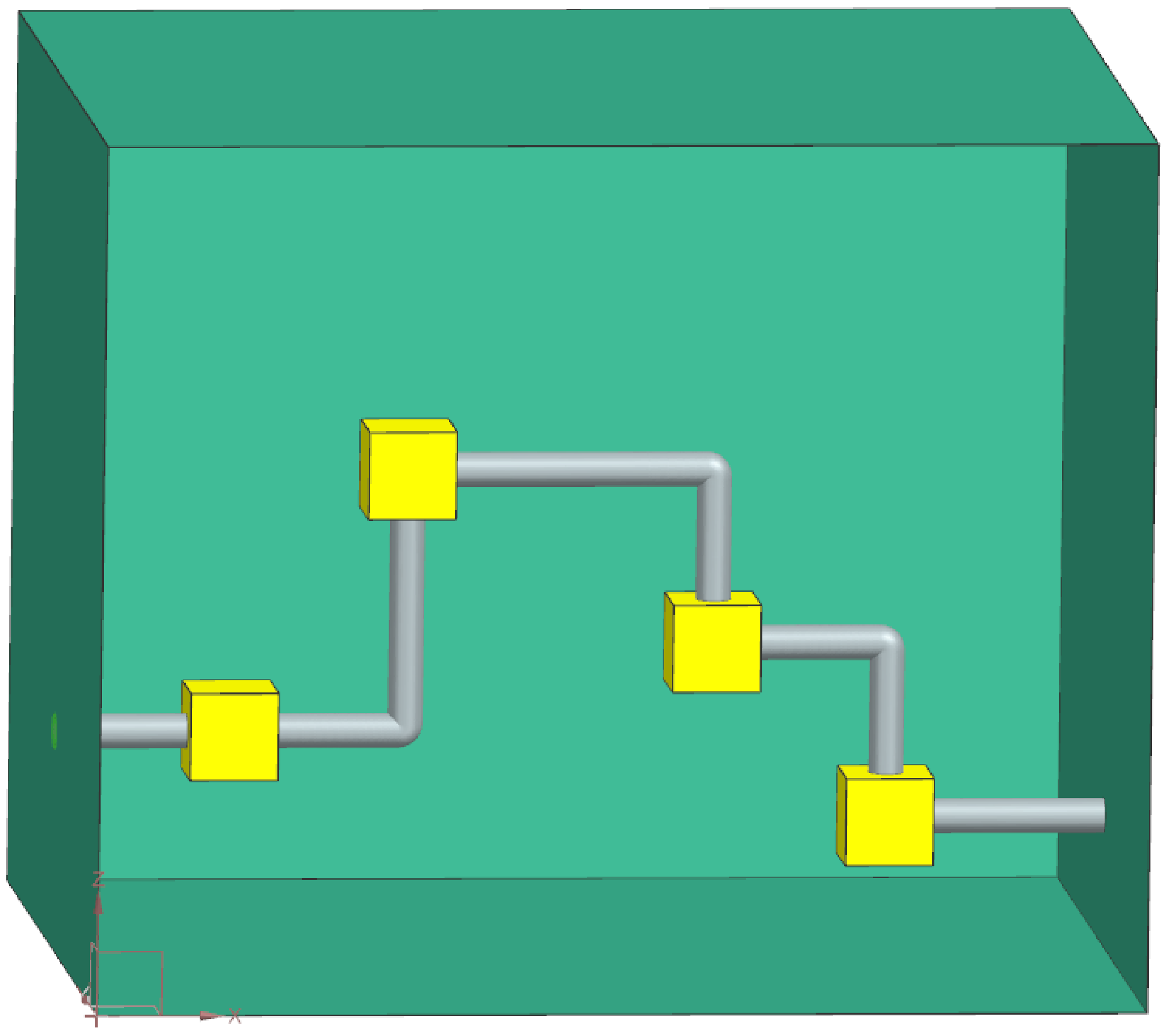
Example 1: Two pieces of equipment

Point A and B in the environment are placed in the floor and the right wall, respectively.



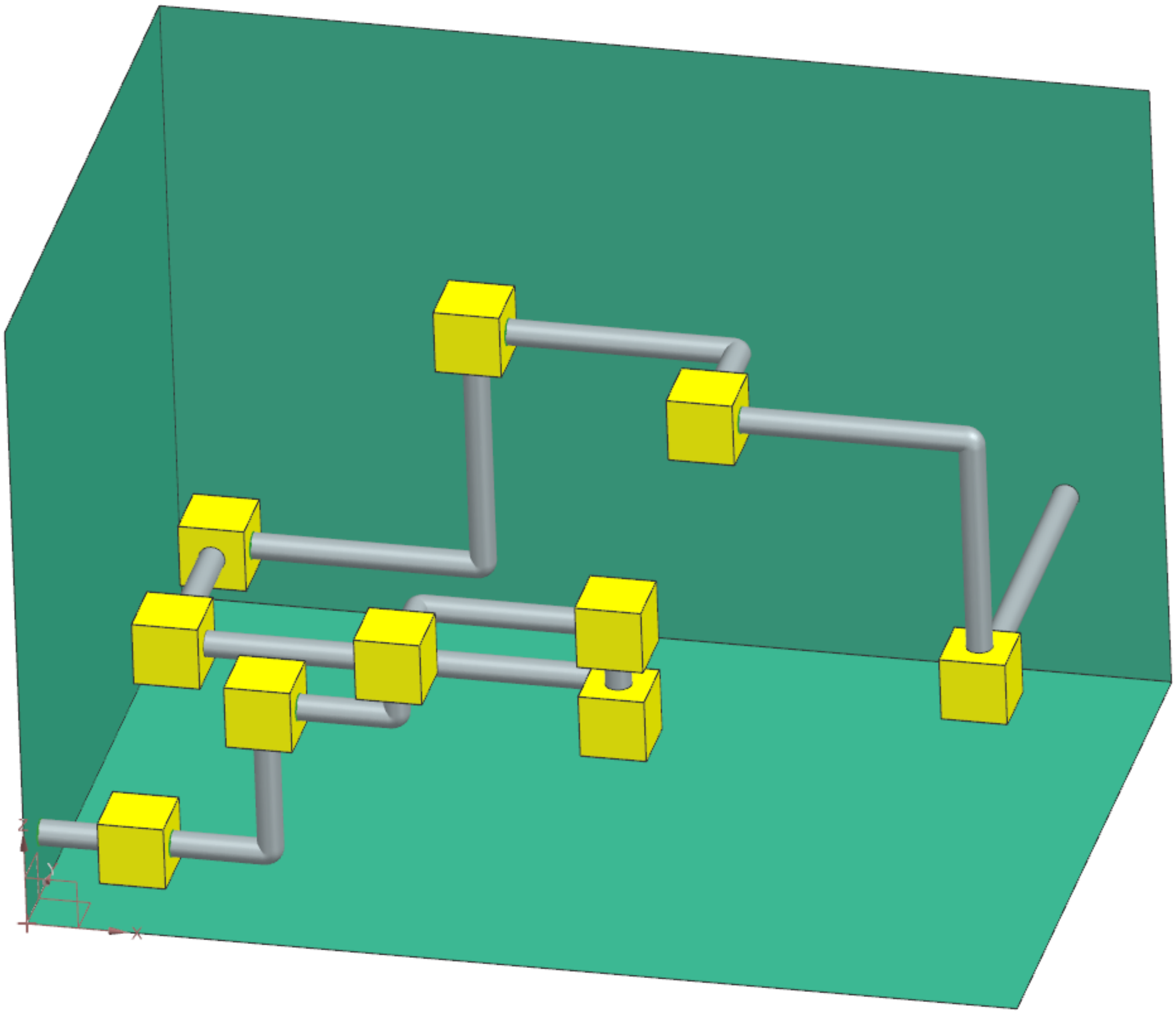
Example 2: Four pieces of equipment

Point A and B in the environment are placed in the left and the right wall, respectively.



Example 3: Ten pieces of equipment

Point A and B in the environment are placed in the left and the back wall, respectively.



Backlog

Included for the purpose of keeping track of some of the future opportunities for the application. List is not exhaustive.

Knowledge base

- Equipment can be defined with a multitude of parameters and stored for later use. Different equipment could entail having to use other equipment, and could be implemented in a way such as “if this is used, we need either this or this as well, or the environment temperature needs to be such and such”.
- ParameterContainer would then probably be obsolete.

Diameter decision making

- Make it so that the piping diameter is dimensioned from what the equipment with the largest input/output diameter is.

Obstacles

- Add obstacles to the equation.
- Pipes: create attachment plates with bolt holes for the in and outlets.

Pipes are plummed

- Opening them up was harder than imaginged when the pipe makes a turn.

The travelling salesman problem

- The path taken between equipment should be calculated as a solution to the travelling salesman problem, and not solely dependent on the order in which the equipment is fed to the application as it is today.
- There are multiple algorithms for solving this problem. The algorithm for this application needs not to be exact.