MAINFORMATICS

A Study on Automated Fault Tree Generation Using Object-Oriented Programing of Simplified P&ID Components

Jinok Lee, Ho Seok, Gyunyoung Heo*

한국원자력학회 2025추계학술발표회 및 제58회 정기총회
2025년 10월 29일(수) ~ 31일(금)
창원컨벤션센터

KHU

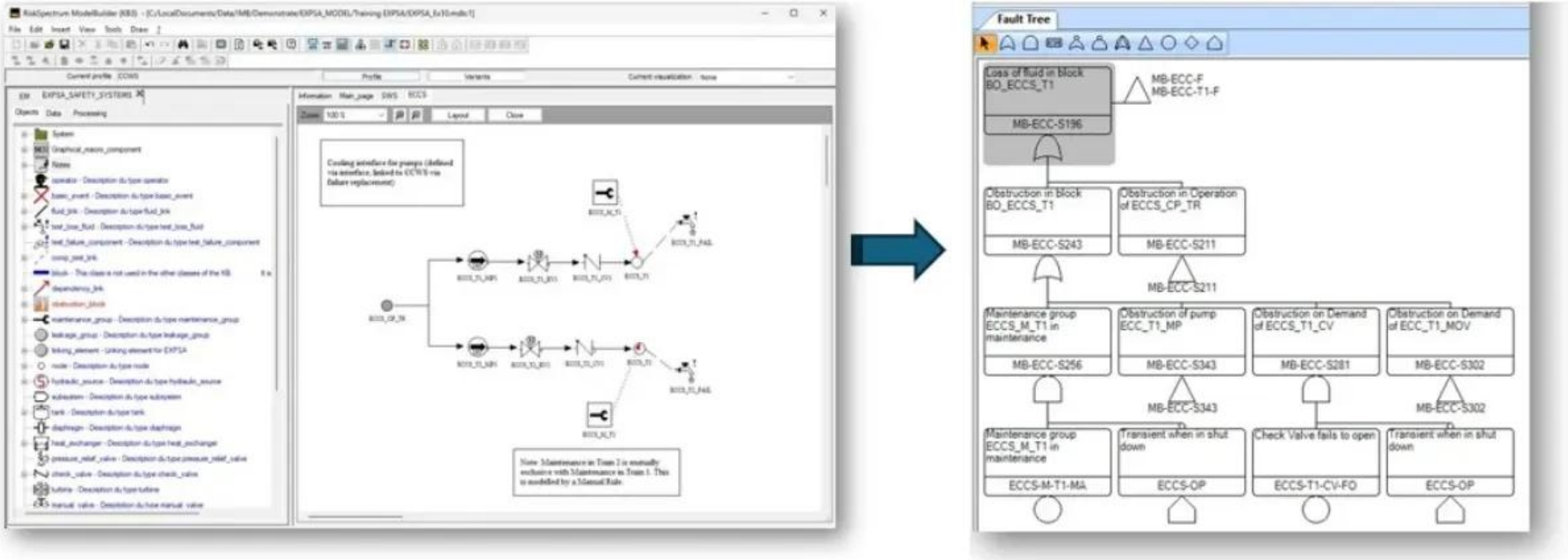
Table of Contents

I. Introduction

- Development of advanced nuclear reactors increases the importance of Probabilistic Safety Assessment (PSA) from the early design stage.
- Effective PSA modeling requires close collaboration between designers and PSA practitioners.
- In early design phases, the ability to quickly construct and update PSA models is more critical than building highly detailed ones.
- PSA includes various technical elements, among which Fault Tree Analysis (FTA) is considered most suitable for automation in the design process.
- Manual FTA approaches cause:
  - Inconsistency in analysis
  - Time and cost inefficiency
  - Higher error potential
- In the past, Simplified P&IDs existed mainly as paper documents or unstructured digital formats (e.g., PDFs), making automatic Fault Tree (FT) generation impossible.
- Recently, database-based design environments have enabled P&IDs to be created in structured, digital formats, providing new opportunities for automation.

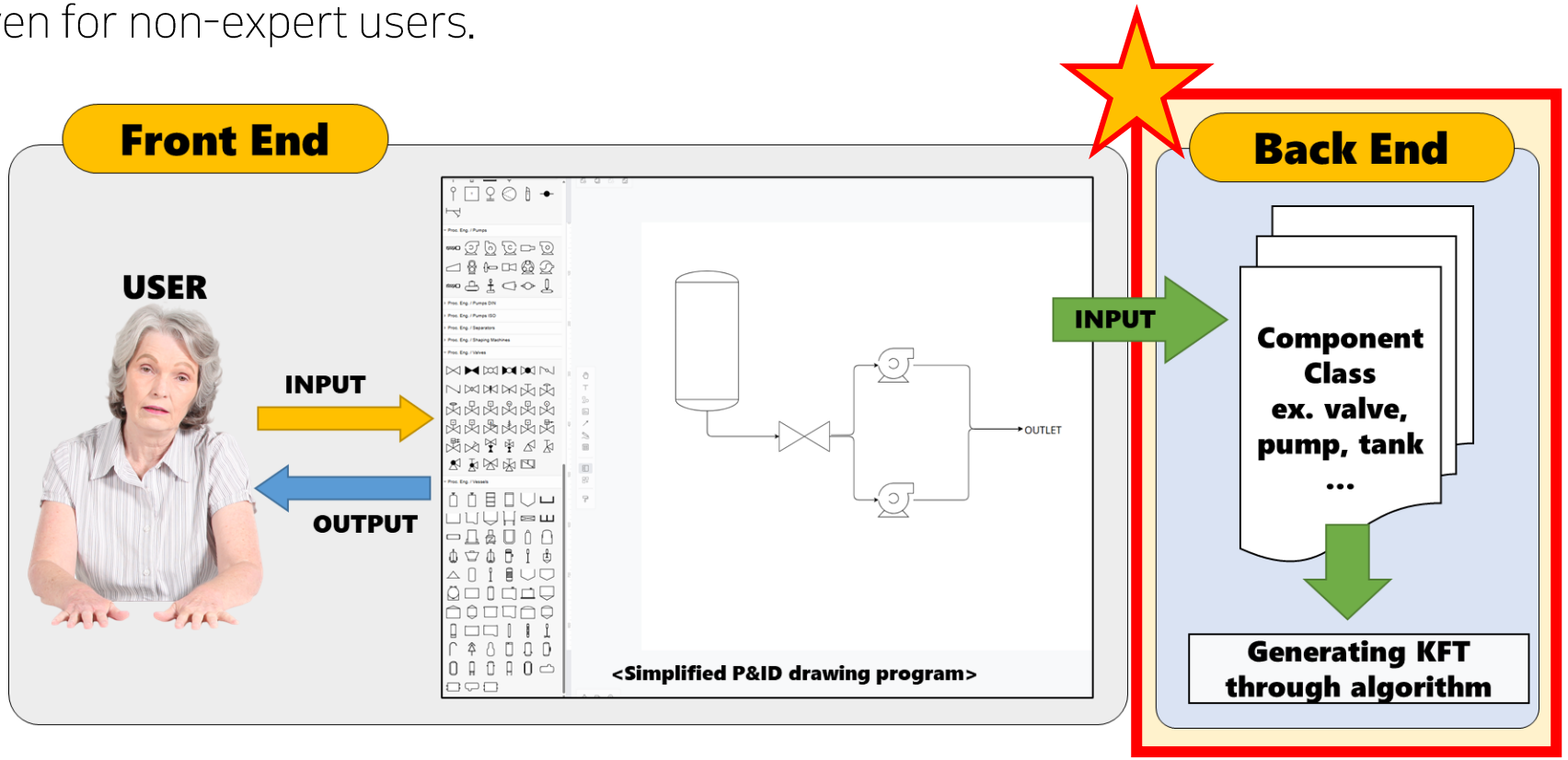
I. Introduction

- RiskSpectrum ModelBuilder



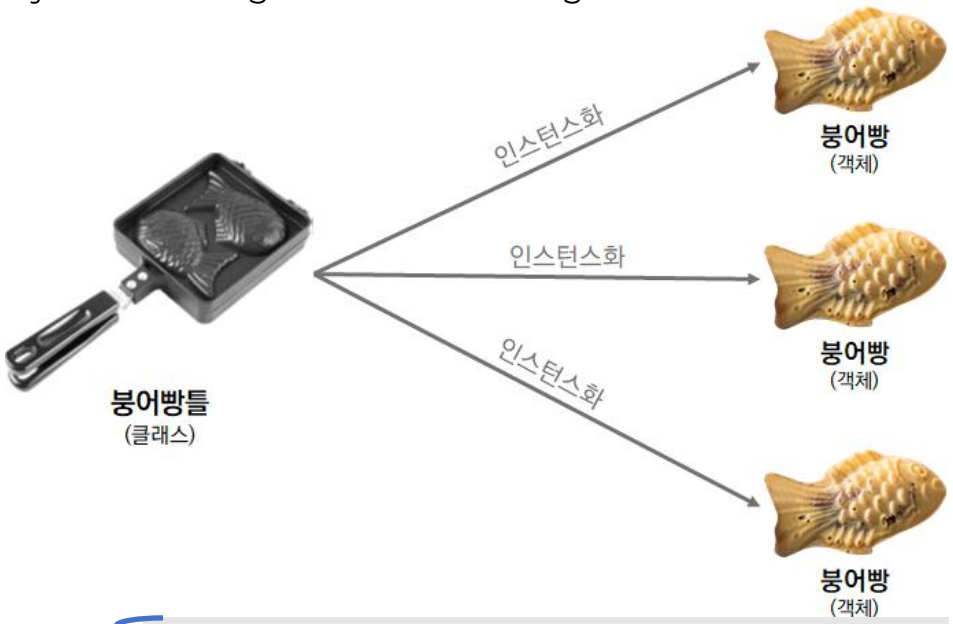
I. Introduction

- The objective of this study is to develop an automated tool that converts P&ID to Fault Tree (FT).
- The tool aims to replace the existing inefficient manual process, enabling faster, easier, and more accessible FT generation, even for non-expert users.



II. Research Methodology

- What is object-oriented programming?
  Object-Oriented Programming (OOP) is a programming paradigm in which data is organized and managed as "objects." Programs are designed so that these objects interact with each other to perform tasks.




```
bungbbang = object(
        name="붕어빵",
        flavor="팥",
        price="1000"
        ex_data="2025.08.24"
    )
```

Object

Attributes

```
baking_bungfish(bungbbang)
```

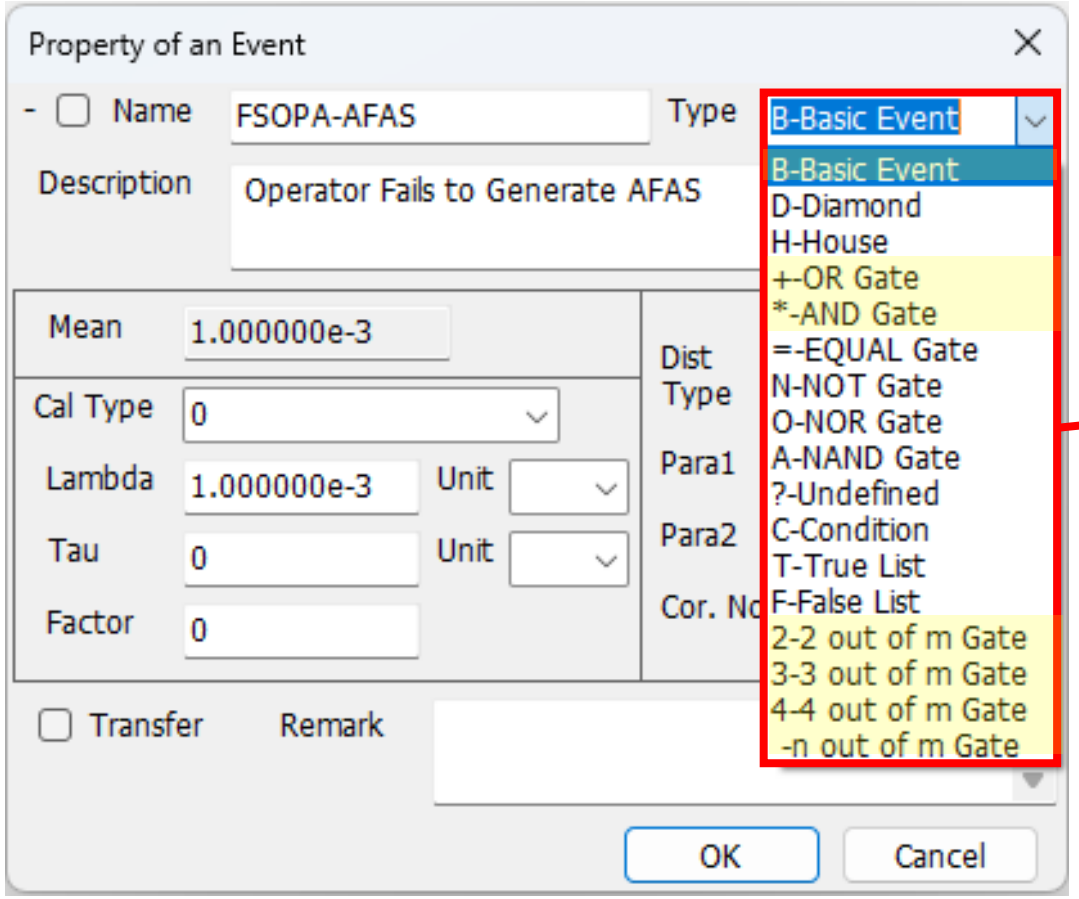
Methods

Class

```
class bungfish:
    name: str = "",
    flavor: tr = "",
    price: int
    ex_data: datetime
```

II. Research Methodology

- Component class definition



Attribute Name	Description
no	Unique component number
id	Unique ID (Primary Key)
parent	List of parent components (Foreign Key)
child	List of child components (Foreign Key)
type	**Logic type (e.g., B, +, *)**
component	Equipment type (e.g., Valve, Pump)
status	Status information (e.g., open, stop)
capacity	Design capacity
ccf	Indicates Common Cause Failure (True = CCF)
ccfID	Related CCF component ID
exception	Indicates exclusion from analysis (True = excluded)

***System-defined Attributes, User-defined Attributes,* Auto**

II. Research Methodology

- Pseudo-code for FT Generation

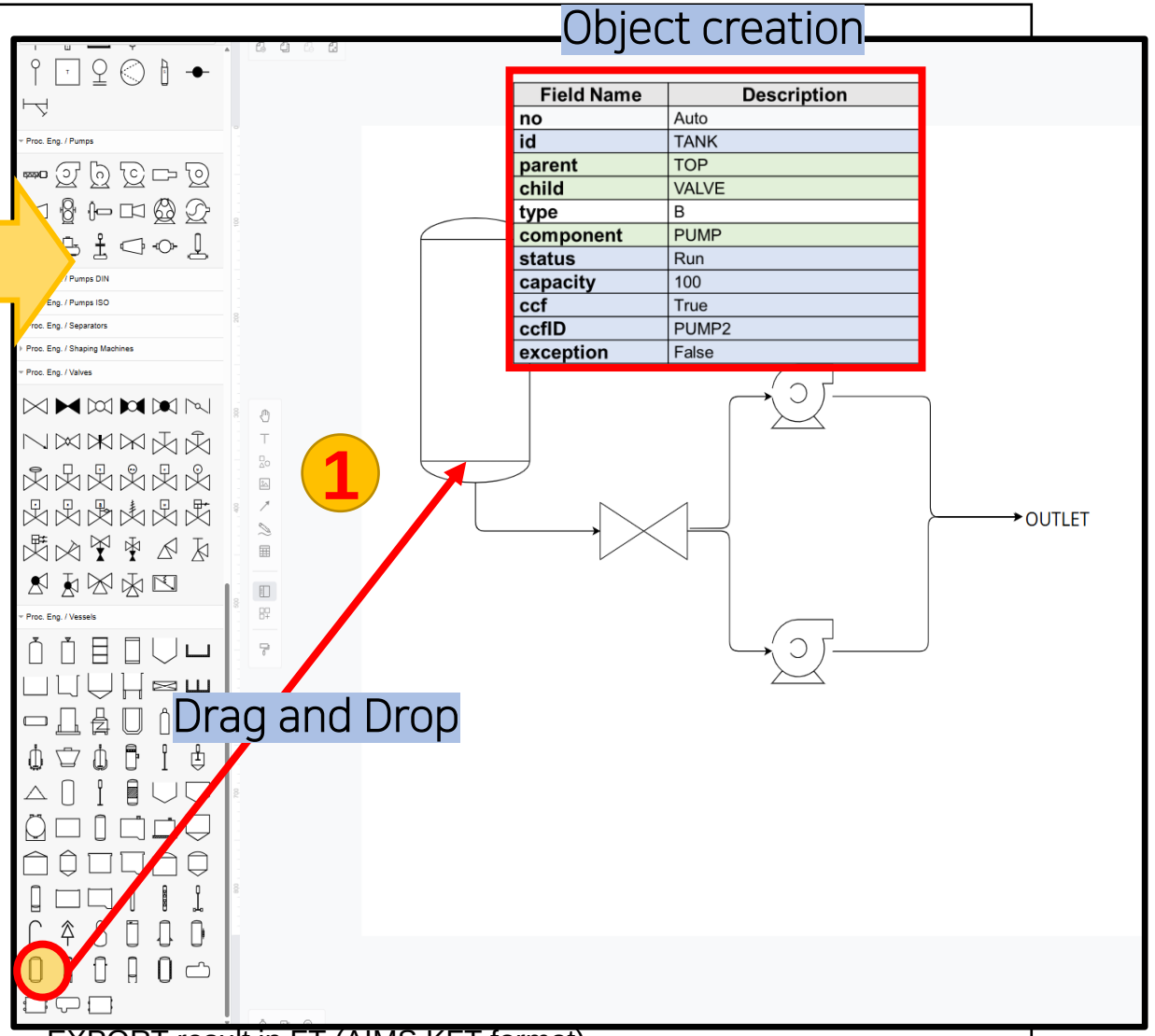
```
# Step 1. Component Object Creation
FOR each component ADDED in Simplified P&ID:
    CREATE new ComponentClass
    user DEFINES component.id
    component.component ← automatically assigned (based on type: valve, pump, etc.)
    component.status ← automatically assigned (based on current state)

# Step 2. Node Connection Definition
IF user CONNECTS nodes:
    UPDATE component.parent
    UPDATE component.child

# Step 3. User-Defined Fields
FOR each component:
    user INPUTS component.capacity
    user INPUTS component.status
    IF component is in a CCF relation:
        component.ccf ← True
        component.ccfID ← related component.id

# Step 4. Database-Based Component Handling
    IF component.component EXISTS in Database:
        CREATE gate
        CREATE basic events (based on predefined failure modes)

# Step 5. Gate Generation Rules
FOR each component:
    IF component.parent COUNT ≥ 2:
        total_capacity ← SUM(child.capacity)
        IF total_capacity > 100:
            CREATE AND gate
        ELSE:
            CREATE OR gate
        IF child COUNT > 2:
            K ← COUNT(combinations of child where SUM(capacity) > 100)
            N ← child COUNT
            CREATE K-out-of-N AND gate

# Step 6. CCF Basic Event Generation
IF component.ccf == True:
    component.ccfID ← check
    CREATE basic events (based on ccf relation)


# Step 7. Output Generation
IF all gates and basic events are defined:
    PERFORM logic operations
    EXPORT result in FT (AIMS KFT format)
```

III. Case Study

 # Step 1. Component Object Creation
FOR each component ADDED in Simplified P&ID:
CREATE new ComponentClass
user DEFINES component.id
component.component ← automatically assigned (based on type: valve, pump, etc.)
component.status ← automatically assigned (based on current state)

 # Step 2. Node Connection Definition
IF user CONNECTS nodes:
UPDATE component.parent
UPDATE component.child

 # Step 3. User-Defined Fields
FOR each component:
user INPUTS component.capacity
user INPUTS component.status
IF component is in a CCF relation:
component.ccf ← True
component.ccfID ← related component.id



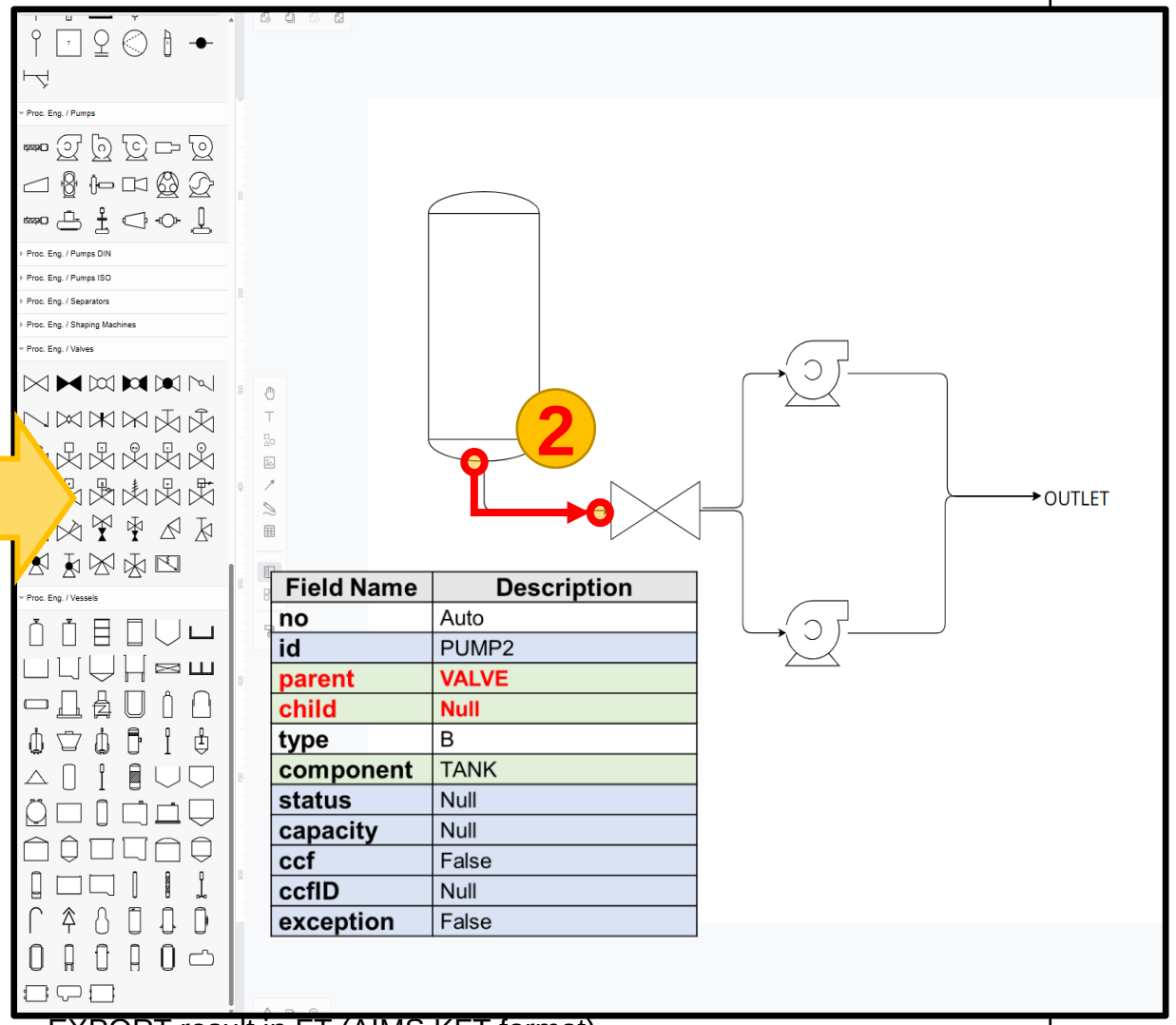
EXPORT result in FT (AIMS KFT format)

III. Case Study

 # Step 1. Component Object Creation
FOR each component ADDED in Simplified P&ID:
    CREATE new ComponentClass
    user DEFINES component.id
    component.component ← automatically assigned (based on type: valve, pump, etc.)
    component.status ← automatically assigned (based on current state)

 # Step 2. Node Connection Definition
IF user CONNECTS nodes:
    UPDATE component.parent
    UPDATE component.child

 # Step 3. User-Defined Fields
FOR each component:
    user INPUTS component.capacity
    user INPUTS component.status
    IF component is in a CCF relation:
        component.ccf ← True
        component.ccfID ← related component.id



Field Name	Description
no	Auto
id	PUMP2
parent	VALVE
child	Null
type	B
component	TANK
status	Null
capacity	Null
ccf	False
ccfID	Null
exception	False

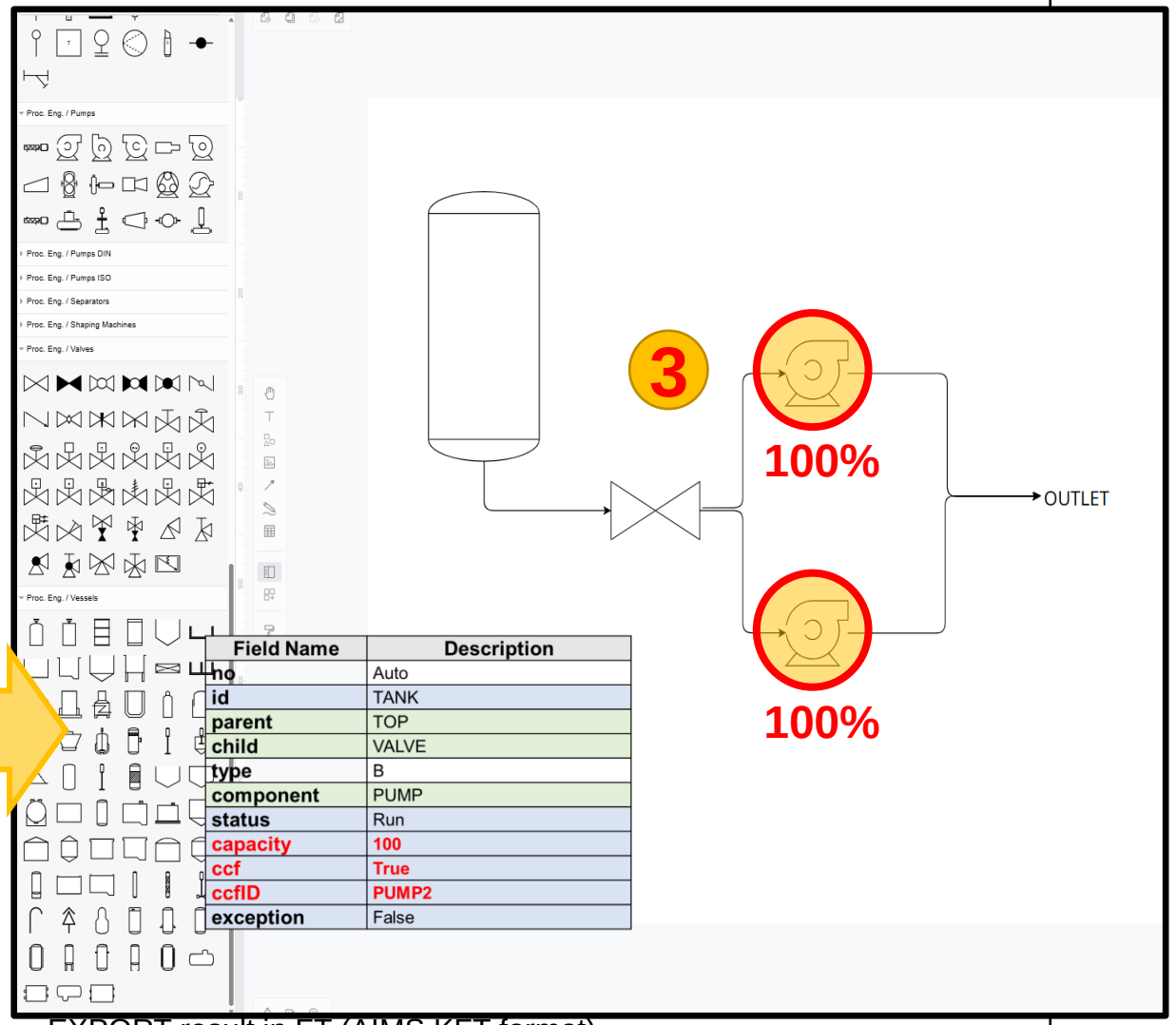
EXPORT result in FT (AIMS KFT format)

III. Case Study

 # Step 1. Component Object Creation
FOR each component ADDED in Simplified P&ID:
    CREATE new ComponentClass
    user DEFINES component.id
    component.component ← automatically assigned (based on type: valve, pump, etc.)
    component.status ← automatically assigned (based on current state)

 # Step 2. Node Connection Definition
IF user CONNECTS nodes:
    UPDATE component.parent
    UPDATE component.child

 # Step 3. User-Defined Fields
FOR each component:
    user INPUTS component.capacity
    user INPUTS component.status
    IF component is in a CCF relation:
        component.ccf ← True
        component.ccfID ← related component.id

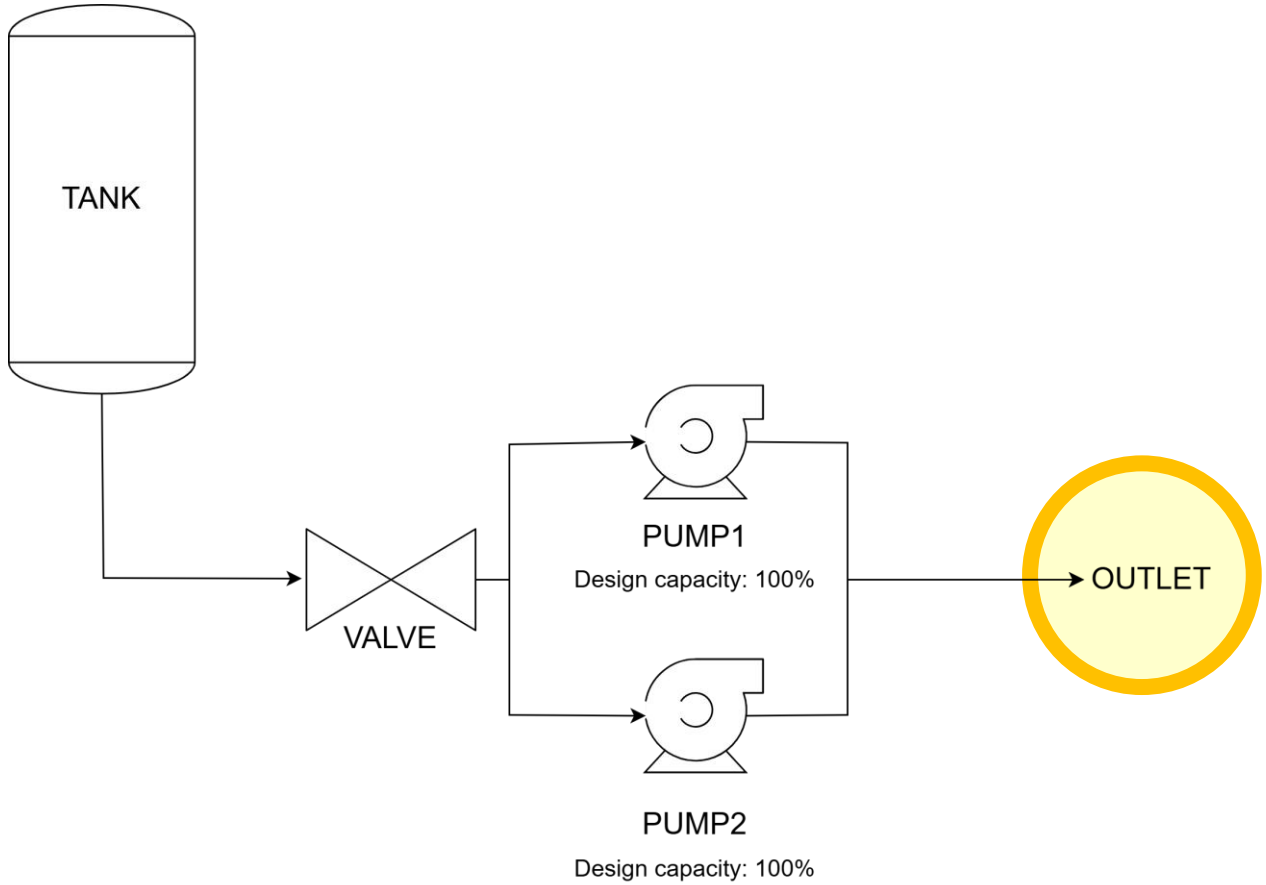


Field Name	Description
no	Auto
id	TANK
parent	TOP
child	VALVE
type	B
component	PUMP
status	Run
capacity	100
ccf	True
ccfID	PUMP2
exception	False

EXPORT result in FT (AIMS KFT format)

III. Case Study

- Example of a Component Object – TOP

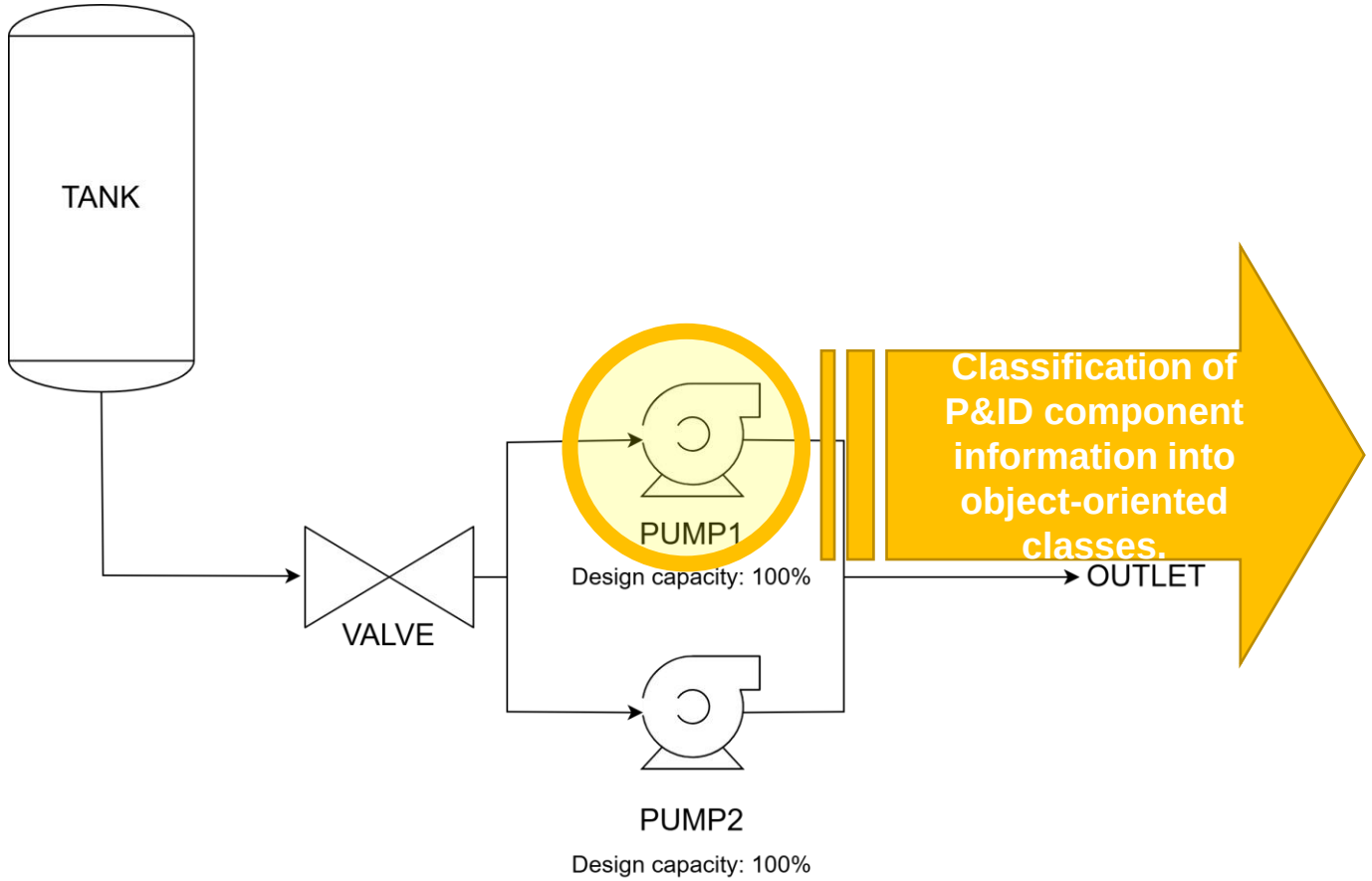


Field Name	Description
no	Auto
id	TOP
parent	Null
child	TANK
type	*
component	Null
status	TOP
capacity	Null
ccf	False
ccfID	Null
exception	False

System-defined Attributes,** **User-defined Attributes,** **Auto

III. Case Study

- Example of a Component Object – PUMP1

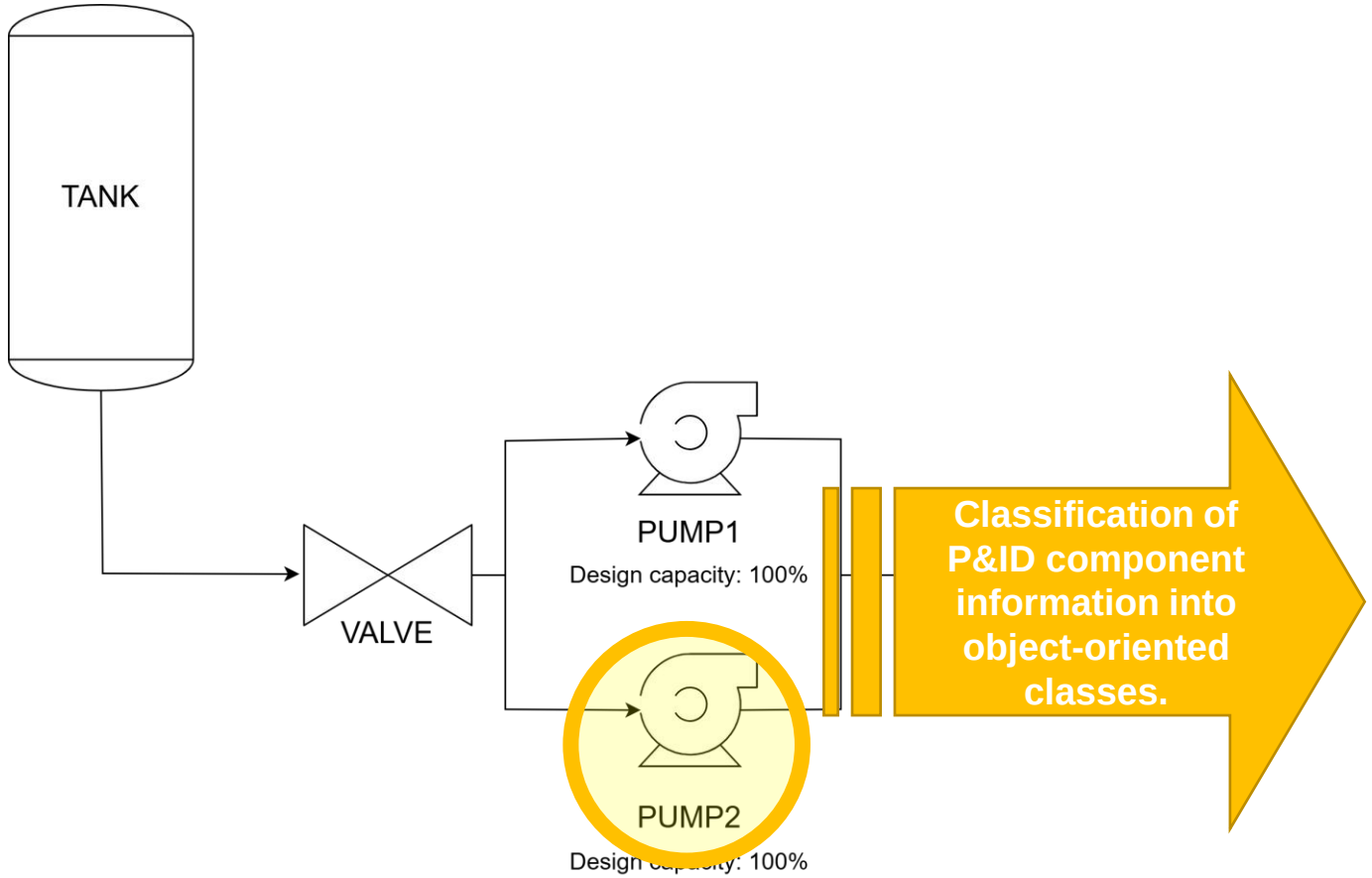


Field Name	Description
no	Auto
id	TANK
parent	TOP
child	VALVE
type	B
component	PUMP
status	Run
capacity	100
ccf	True
ccfID	PUMP2
exception	False

System-defined Attributes, User-defined Attributes, Auto

III. Case Study

- Example of a Component Object – PUMP2

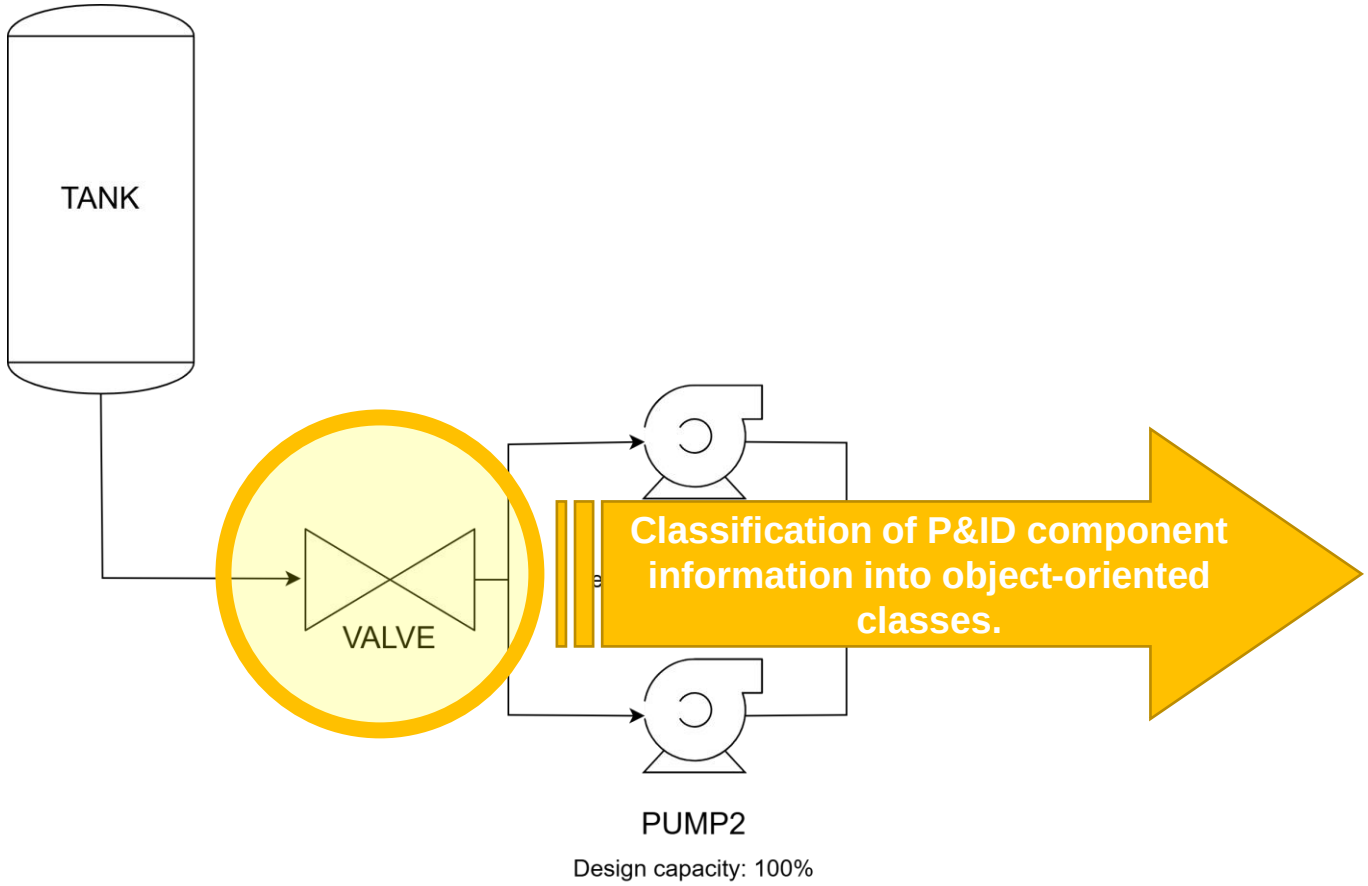


Field Name	Description
no	Auto
id	VALVE
parent	TOP
child	VALVE
type	B
component	PUMP
status	Run
capacity	100
ccf	True
ccfID	PUMP1
exception	False

System-defined Attributes, User-defined Attributes, Auto

III. Case Study

- Example of a Component Object – VALVE

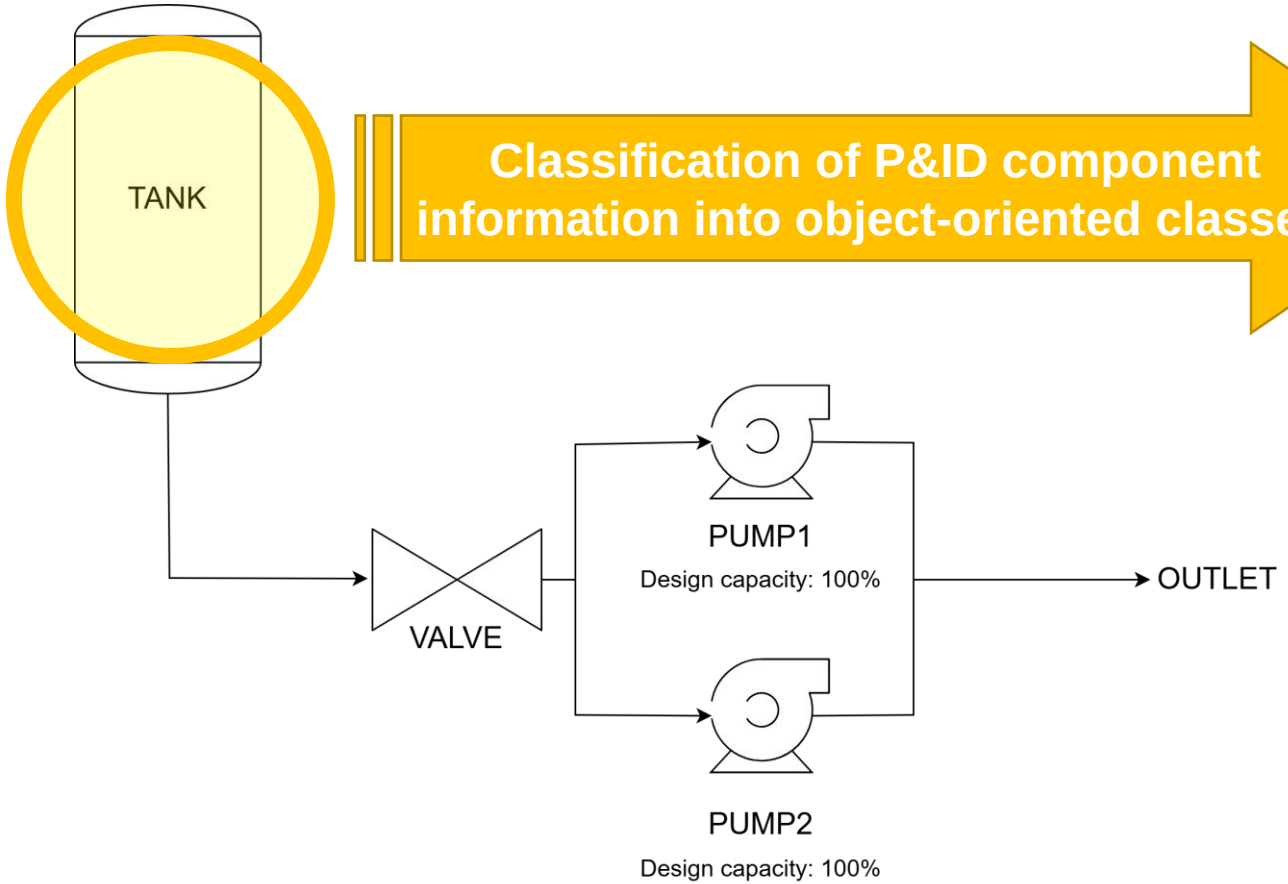


Field Name	Description
no	Auto
id	PUMP1
parent	PUMP1PUMP2
child	TANK
type	B
component	VALVE
status	OPEN
capacity	Null
ccf	False
ccfID	Null
exception	False

System-defined Attributes, User-defined Attributes, Auto

III. Case Study

- Example of a Component Object – TANK



Field Name	Description
no	Auto
id	PUMP2
parent	VALVE
child	Null
type	B
component	TANK
status	Null
capacity	Null
ccf	False
ccfID	Null
exception	False

System-defined Attributes, User-defined Attributes, Auto

III. Case Study

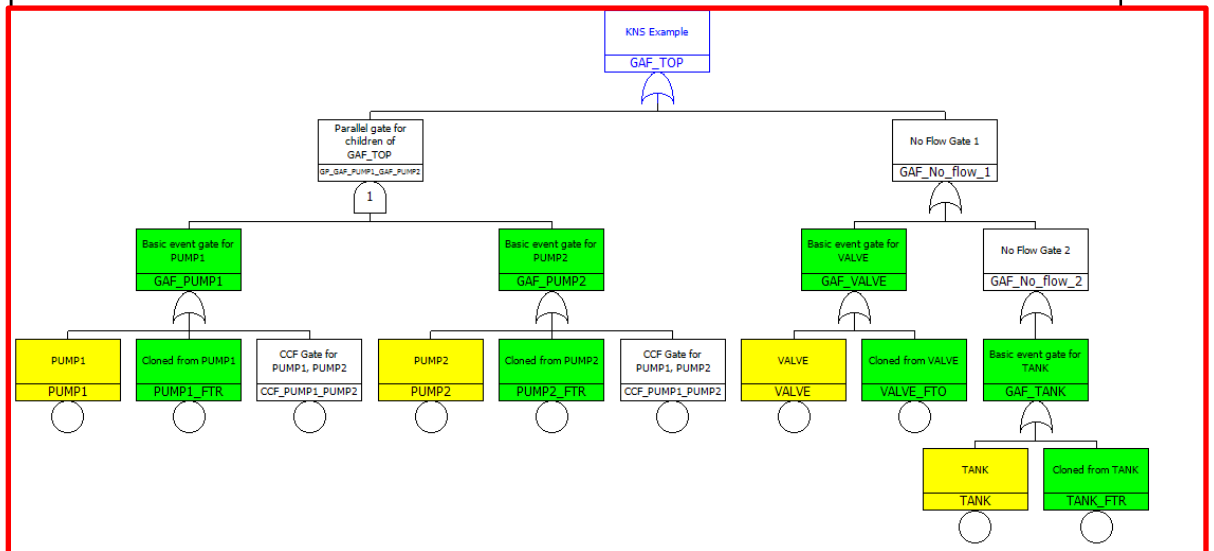
Class Field	TOP EVENT	PUMP1	PUMP2	VALVE	TANK
no	Auto	Auto	Auto	Auto	Auto
id	TOP	TANK	VALVE	PUMP1	PUMP2
parent	Null	TOP	TOP	PUMP1 PUMP2	VALVE
child	PUMP1 PUMP2	VALVE	VALVE	TANK	Null
t	*	B	B	B	B
component	Null	PUMP	PUMP	VALVE	TANK
status	TOP	Run	Run	OPEN	NULL
capacity	Null	100	100	Null	Null
ccf	False	True	True	False	False
ccfID	Null	PUMP2	PUMP1	Null	Null
exception	False	False	False	False	False

4

System-defined Attributes,** **User-defined Attributes,** **Auto

 # Step 4. Database-Based Component Handling

IF component.component EXISTS in Database:
CREATE gate
CREATE basic events (based on predefined failure modes)



IF component.ccf == True:
component.ccfID ← check
CREATE basic events (based on ccf relation)

 # Step 7. Output Generation

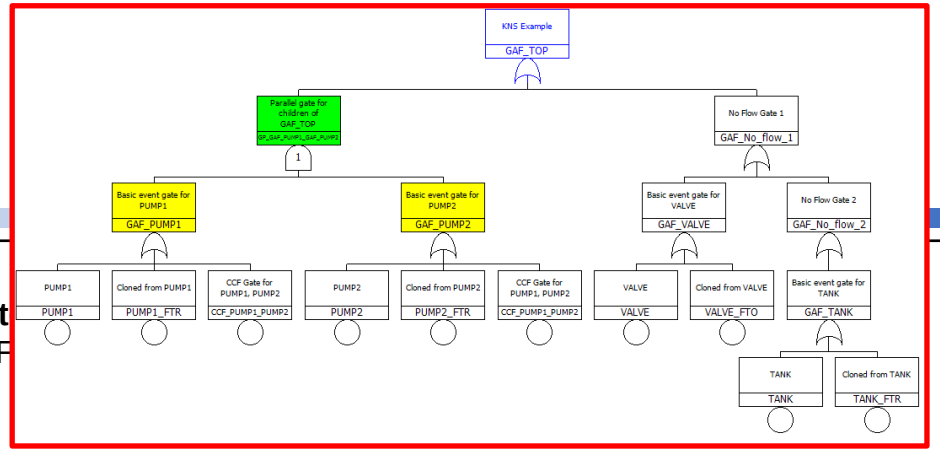
IF all gates and basic events are defined:
PERFORM logic operations
EXPORT result in FT (AIMS KFT format)

III. Case Study

Class Field	TOP EVENT	PUMP1	PUMP2	VALVE	TANK
no	Auto	Auto	Auto	Auto	Auto
id	TOP	TANK	VAL	PUMP1	PUMP2
parent	Null	TOP	TOP	PUMP1 PUMP2	VALVE
child	PUMP1 PUMP2	VALVE	VALVE	TANK	Null
type	*	B	B	B	B
component	Null	PUMP	PUMP	VALVE	TANK
status	TOP	Run	Run	OPEN	NULL
capacity	Null	100	100	Null	Null
ccf	False	True	True	False	False
ccfID	Null	PUMP2	PUMP1	Null	Null
exception	False	False	False	False	False

5

System-defined Attributes,** **User-defined Attributes,** **Auto



 # St
IF

 # Step 5. Gate Generation Rules
FOR each component:
  IF component.parent COUNT ≥ 2:
    total_capacity ← SUM(child.capacity)
    IF total_capacity > 100:
      CREATE AND gate
    ELSE:
      CREATE OR gate
    IF child COUNT > 2:
      K ← COUNT(combinations of child where SUM(capacity) > 100)
      N ← child COUNT
      CREATE K-out-of-N AND gate

 # Step 6. CCF Basic Event Generation
IF component.ccf == True:
  component.ccfID ← check
  CREATE basic events (based on ccf relation)

 # Step 7. Output Generation
IF all gates and basic events are defined:
  PERFORM logic operations
  EXPORT result in FT (AIMS KFT format)

III. Case Study

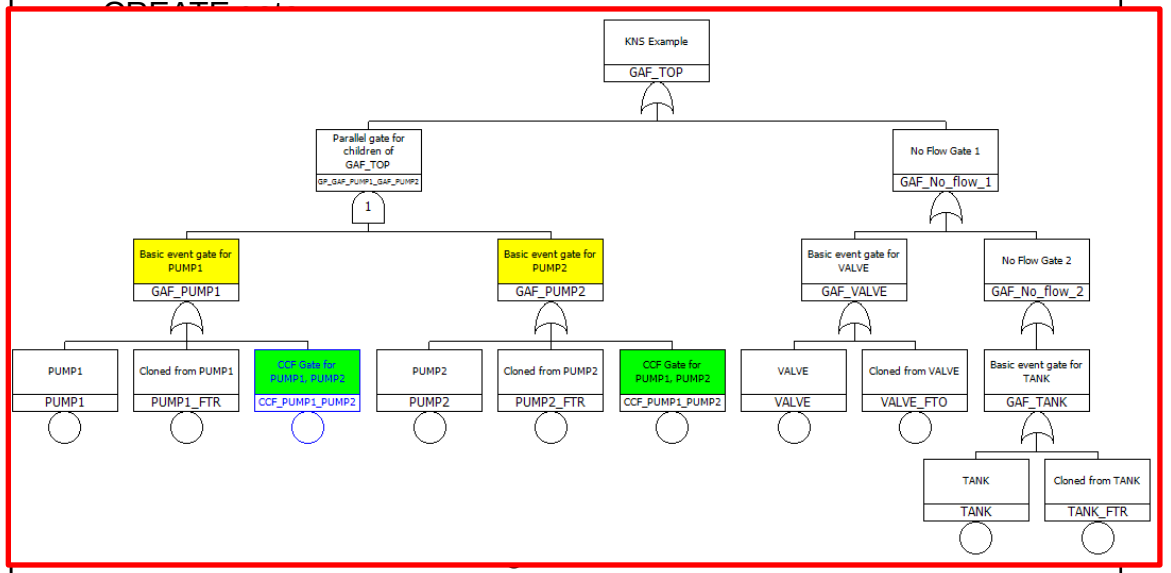
Class Field	TOP EVENT	PUMP1	PUMP2	VALVE	TANK
no	Auto	Auto	Auto	Auto	Auto
id	TOP	TANK	VALVE	PUMP1	PUMP2
parent	Null	TOP	TOP	PUMP1 PUMP2	VALVE
child	PUMP1 PUMP2	VALVE	VALVE	TANK	Null
type	*	B	B	B	B
component	Null	PUMP	PUMP	VALVE	TANK
status	TOP	Run	Run	OPEN	NULL
capacity	Nu	100	100	Null	Null
ccf	False	True	True	False	False
ccfID	Null	PUMP2	PUMP1	Null	Null
exception	False	False	False	False	False

6

System-defined Attributes,** **User-defined Attributes,** **Auto

 # Step 4. Database-Based Component Handling

IF component.component EXISTS in Database:



 # Step 6. CCF Basic Event Generation

IF component.ccf == True:

component.ccfID ← check

CREATE basic events (based on ccf relation)

 # Step 7. Output Generation

IF all gates and basic events are defined:

PERFORM logic operations

EXPORT result in FT (AIMS KFT format)

III. Case Study

```
"#KIRAP_TREE Version 3.5"
"Title=",""
"UserName=","isa"
"DataFileName=",""
"RecoveryFileName=",""
"Comments=",""
"#NoXEvent=",20
"#XEventData"
1,"GAF_TOP","+",0,0
0,0,0.0,0,0,"L",0,"","",0.0,0,0
"KNS Example"
""
2,"GP_GAF_PUMP1_GAF_PUMP2_GAF_PUMP3","2",0,0
0,0,0.0,0
"Parallel
""
3,"GAF_No_flow_1","+",0,0
0,0,0.0,0,0,"L",0,"","",0.0,0,0
"No Flow Gate 1"
""
4,"CCF_PUMP1_PUMP2_PUMP3","B",0,0
0,0,0.0,0,0,"L",0,"","",0.0,0,0
"CCF Gate for PUMP1, PUMP2, PUMP3"
""
5,"GAF_No_flow_2","+",0,0
0,0,0.0,0,0,"L",0,"","",0.0,0,0
"No Flow Gate 2"
""
```

7 Generate .kft file

 # Step 4. Parent-Child Re-definition
FOR each component WHERE component.child IS NULL:
  TRACE parent upward
  IF parent.type == Basic Event:
    REDEFINE parent field to reference the higher-level component

 # Step 5. Gate Generation Rules
FOR each component:
  IF component.child COUNT ≥ 2:
    total_capacity ← SUM(child.capacity)

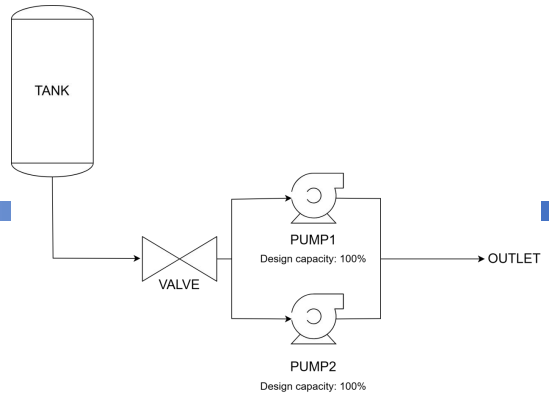
    IF total_capacity > 100:
      CREATE AND gate
    ELSE:
      CREATE OR gate

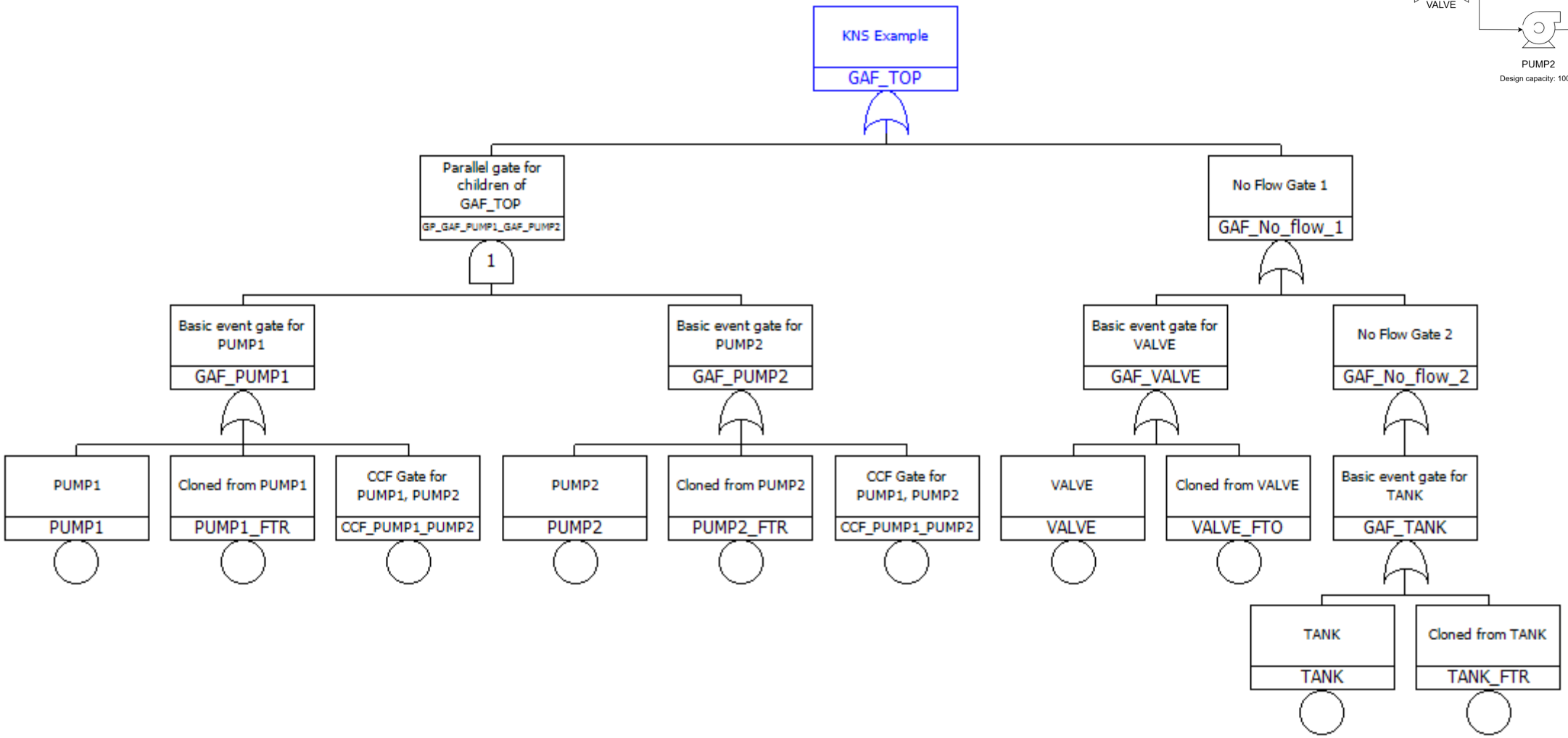
    IF child COUNT > 2:
      K ← COUNT(combinations of child where SUM(capacity) > 100)
      N ← child COUNT
      CREATE K-out-of-N AND gate

 # Step 6. Database-Based Component Handling
  IF component.component EXISTS in Database:
    CREATE gate
    CREATE basic events (based on predefined failure modes)

 # Step 7. Output Generation
IF all gates and basic events are defined:
  PERFORM logic operations
  EXPORT result in FT (AIMS KFT format)

III. Case Study





IV. Conclusions

- Functional Requirements

P&ID Drawing Tool	Algorithm
1. Provides the necessary components for drawing P&ID diagrams. components (e.g., valves, tanks, pumps)	1. Gates must be generated based on the attribute values of component objects
2. When a component is placed on the drawing, an object should be created automatically and its fields auto-filled.	2. The database must define the default Basic Events for each component.
3. The tool should allow users to define connections between objects using nodes.	3. The algorithm should identify CCF relationships and generate CCF Basic Events.
4. An input window should be provided so users can define or edit fields as needed.	4. Gates should be generated based on parallel relationships between objects.
	5. The fault tree must be generated by analyzing the connections between all component objects.

IV. Conclusions

- Future Plan

P&ID Drawing Tool	Algorithm
1. Upgrade the user interface (e.g., allow CCF relationships to be defined using keyboard shortcuts).	1. Apply appropriate naming rules (currently set arbitrarily).
2. Incorporate additional guidelines from the Fault Tree Handbook.	2. Automatically generate appropriate descriptions (currently set arbitrarily).
3. Input fields for entering the parameters required by AIMS PSA	3. Create a failure rate database (default is 6928).
4.	4. Automatically generate gates based on success criteria.
5.	5. Expand failure modes according to component types.
6.	6.
7.	7.
8.	8.
9.	9.

IV. Conclusions

- This study proposes a methodology for automatic Fault Tree (FT) generation based on Simplified P&ID.
- Key components such as tanks, valves, and pumps were modeled using an object-oriented class structure to define their attributes, states, performance, and hierarchical relationships systematically.
- Based on this structure, an algorithm was developed to construct standardized FT generation procedures and consistent hierarchical logic automatically.
- The current framework requires manual input of CCF relationships; the algorithm cannot yet identify them autonomously.
- Future work will focus on enabling automatic CCF detection and event generation directly from Simplified P&ID data.
- Additional validation is needed for complex system applications, as the current study used a simplified case.

Reference

[1] Arroyo, Esteban, et al. "Automatic derivation of qualitative plant simulation models from legacy piping and instrumentation diagrams." Computers & Chemical Engineering 92 (2016): 112-132.
[2] Bäckström, Ola, et al. "Flexibility of analysis through knowledge bases." Proc. 31th Eur. Saf. Rel. Conf. 2021.
[3] U.S. Nuclear Regulatory Commission (NRC). "Fault Tree Handbook." NUREG-0492, Washington, DC, 1981.

MAINFORMATICS

My Git Hub

한국원자력학회 2025추계학술발표회 및 제58회 정기총회

Presenter : Jinok Lee/ wlsdhr6615@khu.ac.kr

Corresponding Author : Gyunyong Heo / gheo@khu.ac.kr

KHU