

# Verification of Process Plant Models

Claire Palmer and Paul W.H. Chung

Department of Chemical Engineering  
Loughborough University  
Ashby Road  
Loughborough  
Leicestershire, LE11 3TU, U.K.  
C.Palmer@lboro.ac.uk  
P.W.H.Chung@lboro.ac.uk

## Abstract

The motivation for creating and verifying process plant models is briefly described. Verification is defined. Verification techniques for signed directed graphs models of process plants are detailed. Two case studies are presented to evaluate the verification techniques. A limitation with the verification is detailed.

## Motivation

The identification of possible hazards in chemical plants is a very important part of the design process. This is because of the potential danger that large chemical installations pose to the public. One possible route for speeding up the identification of hazards in chemical plants is to use computers to identify hazards automatically. This will facilitate safe plant design and will avoid late design changes which can be very costly to implement.

Previous research at Loughborough has concentrated on developing a model-based approach and a fault propagation analysis algorithm for automating hazard identification, resulting in the expert system QUEEN (Qualitative Effects Engine) (Chung, 1993). The results generated have demonstrated the technical feasibility of the approach. QUEEN uses unit-based signed directed graph models to perform qualitative analysis of the effects of process deviations in process plants.

A signed directed graph (Iri et al., 1979; Oyeleye and Kramer, 1988; Wilcox and Himmelblau, 1994; Chung, 1993; Larkin et al., 1997) consists of an influence graph with labelled arcs. An influence graph contains the variables in a physical system which are depicted as nodes. These are connected by arcs to reflect the influence the variables have on one another. An arc from a node X, to another node Y, indicates that a change in the variable X will cause a change in the variable Y. An influence is defined as a causal relation between two variables or simply how one variable affects the other. Several

variations of signed directed graph exist in which the arcs may be labelled differently. In the format used here each arc of the graph is labelled with a sign "+" or "-". The sign "+" indicates a positive influence, i.e. Y will increase if X is increased and Y will decrease if X is decreased. The sign "-" indicates a negative influence, i.e. Y will decrease if X is increased and Y will increase if X is decreased.

To model fault propagation the basic SDG representation is extended by the addition of causes (of deviations) and adverse consequences. The cause and consequence nodes are linked into the deviation network. Cause nodes represent the faults (failure modes) of units. Consequence nodes represent potentially hazardous events arising from causes or deviations. Using this extended representation the node 'X' could be a process variable deviation or a fault. The node 'Y' could be a process variable deviation or a consequence.

To construct an SDG for a whole plant can be very time consuming. However, process plants are built by connecting together a set of smaller units to carry out the required functions. The behaviour of each of these types of units can be modelled generically so that it will apply to any plant in which the unit is used. By combining the unit models the behaviour of the whole plant can be analysed. This unit-based approach is widely used (Chung, 1993; Vaidhyanathan and Ventkatasubramanian, 1995; Catino, Grantham and Ungar, 1991). For the SDG representation each unit model consists of a mini-SDG. The mini-SDG shows how a change in one process variable affects another variable in the same unit. Deviations occurring in the unit can be propagated to other units via inport and outport connections. A SDG for a complete plant is created by joining together the appropriate mini-SDG's based on the plant topology.

A tool, Equipment Model Builder, has been created to build unit-based SDG models (Palmer and Chung 1997; 1998). Applications such as QUEEN require a good library of unit models. All forms of modelling errors occur due to omissions and mistakes. Verification is necessary to detect modelling errors which may give rise to the wrong result when the models are utilised by QUEEN.

## Verification

Verification will be defined here as ensuring that the internal structure of each model is complete, correct and consistent. Verification should also ensure that the model's behaviour is plausible. This means that the model should function correctly. This overlaps with some definitions of validation. For example, Gupta (1993) defines validation as "the functional accuracy or correctness of the system's performance." Ensuring that the internal structure of the model is correct will go some way towards ensuring that the model functions correctly. Some of the techniques used for verification may also be applicable to validation. There is an overlap between verification of a model and validation. However complete validation may only be achieved by testing the model in the environment in which it is going to be used.

The aim of verification is to ensure as far as possible that the model is complete, correct and consistent. In order for a model to be consistent it must be correct and concise.

- A complete model has no missing information. It will contain sufficient information to be able to function in all possible situations that arise in the application. All the information that ought to be in the model is contained in the model structure.
- A correct model is an accurate representation and contains no wrong information. It has no conflicting information or illegitimate attributes. An illegitimate attribute is one which does not occur within the set of attribute values allowed for the model. A correct model will function correctly.
- A concise model has no redundant or duplicated information. It has no information which is superfluous or unnecessary. A model which is not concise will lead to additional processing and may be ambiguous.

## Completeness Techniques

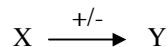
Incompleteness can occur for a number of reasons. The main causes of incompleteness result from:

- unreferenced attributes;
- missing propagation paths.

### Unreferenced Attributes

The term 'unreferenced attributes' has been taken from the body of work describing verification of rule-based systems. Two types of incompleteness check for unreferenced attributes are relevant to this work: the dead-end conclusion check and the unreachable condition check. The dead-end conclusion check identifies rules with conclusions that can never be utilised. The unreachable condition check identifies rules with conclusions whose conditions can never be met.

An SDG may be compared to a rule. Consider the arc shown below.



The node 'X' may be compared to the conditions or left hand side of a rule. The node 'Y' may be compared to the conclusions or right hand side of a rule. The completeness checks for rule-based systems are also applicable to systems constructing SDG models.

How unreferenced attributes may occur in the unit models created by Equipment Model Builder is illustrated using a simple model.

```
frame(tank isa unit,
  [ inports info [ in ],
    outports info [ out ],
    unitports info [ liquid,vapour ],
    propLinks info [

      %propagation
      arc([in,flow],+,[liquid,level]),
      arc([liquid,level],+,[out,flow]),
      arc([in,temp],+,[liquid,temp]),
      arc([liquid,temp],+,[out,temp]),
      arc([vapour,temp],+,[liquid,temp]),

      %faults
      arc([fault,'external fire'],+,[vapour,temp]),

      %consequences resulting from faults
      arc([fault,'leak to environment'],+,[consequence,'loss of material']),

      %consequences resulting from deviations
      arc([deviation,[lessLevel,liquid]],+,[consequence,'vessel emptying']),
      arc([deviation,[moreFlow,out]],+,[consequence,'vessel emptying']),
      arc([deviation,[moreTemperature,liquid]],+,[consequence,'crystallisation'])
    ]
  ]
).
```

It can be seen from the example that there are four types of SDG arc within a unit model:

- deviations linked to deviations, e.g. ([in,temp],+,[liquid,temp]);
- faults linked to deviations, e.g. ([fault,'external fire'],+,[vapour,temp]);
- faults linked to consequences, e.g. ([fault,'leak to environment'],+,[consequence,'loss of material']);
- deviations linked to consequences, e.g. ([deviation,[moreTemperature,liquid]],+,[consequence,'crystallisation'])

The initiating node 'X' of the arc shown could be a deviation or a fault. The influenced node 'Y' could be a deviation or consequence. A consequence may occur directly as a result of a fault (arc type iii). It may also occur as a result of a fault propagating through a deviation or series of deviations. For example, the following fault

propagation might occur within the tank model, leading to the consequence shown:

```
([tank,fault,external fire],+,[tank,vapour,temp])  
([tank,vapour,temp], +, [tank,liquid,temp])  
([tank,deviation,[moreTemperature,liquid]],  
+,[tank,consequence,crystallisation]).
```

The nodes in the path are of the types ii, i and iv respectively. The fault (failure mode) initiating a propagation path need not be present in the same unit as the resulting consequence. The fault may occur in a unit upstream in a plant and cause a deviation which propagates into another unit via its inports, or in a downstream unit and cause a deviation which propagates into another unit via its outports, to result in a consequence. For example, a pipe model linked upstream of the tank model could cause the following consequence in the tank model:

```
([pipe,fault,partly blocked], -, [pipe,out,flow])  
([pipe,out,flow], +, [tank,in,flow])  
([tank,in,flow], +, [tank,liquid,level])  
([tank,deviation,[lessLevel,liquid]],  
+,[tank,consequence,vessel emptying])
```

Thus in order for a fault to have the potential to cause a consequence it must fall into one of the following categories:

1. The fault must be directly linked to the consequence.
2. The fault must cause a deviation which propagates within the unit model to cause a consequence.
3. The fault must cause a deviation which propagates out of the unit model via its inports or outports resulting in a consequence in another unit.

Any fault that does not fall into one of more of these categories is an unreferenced attribute as it has no overall effect. An unreferenced fault (failure mode) causes a deviation with no effect.

Deviations with no effect may be present in fault linked to deviation arcs as described above. They may also be present in deviation linked to deviation arcs in which the deviation does not propagate to a boundary port. Deviations propagating from model boundary ports (ie. in and out ports) are assumed to have an effect as they may propagate out of the unit model resulting in a consequence in another unit. For example, the deviation ‘out,flow’ in the arc ‘([tank,liquid,level],+,[tank,out,flow])’ could propagate out of the pipe unit to cause a potential effect in a downstream unit in a plant.

In order for a consequence to occur it must either:

1. be directly linked to a fault.
2. result from a fault causing a deviation to propagate within a unit model.
3. be linked to the inports or outports of the unit model. This is so that deviations may propagate in to the unit to cause the consequence. An example of this last case for a tank model would be:

```
([tank,deviation,[moreFlow,out]],+,  
[tank,consequence,vessel emptying])
```

Any consequences not fulfilling these criteria are unreferenced attributes as they will never occur. An unreferenced consequence is linked to a deviation without a cause.

Deviations without causes may be present in deviation linked to consequence arcs (described above). They may also be present in deviation linked to deviation arcs in which the deviation does not initiate at a boundary port. Deviations initiating at boundary ports are assumed to have a cause. The cause may propagate into the unit via its boundary ports from elsewhere in the plant. For example, the deviation ‘in1,flow’ in the arc “([tank,in,flow], +, [tank,liquid,level])” may be caused by a fault propagating into the tank unit from an upstream unit when the tank unit is present in a plant description.

For a given model, Equipment Model Builder maintains lists of deviations with no effect and deviations without causes. Each new arc added or deleted is checked to see if it causes these lists to require updating. A user may access these lists from a drop-down menu.

## Missing Propagation Paths

In order for unit models to function correctly within a plant model deviations in process variables will need to be able to propagate through them. This means that process variable deviations need to be able to propagate from a units inports to its outports and from its outports to its inports. Exceptions to this are models for the source and outlet of the plant. Some unit models may not propagate all deviations. For example, an open tank will not propagate an increase in flow from its outport to its inport. However most units will propagate most deviations.

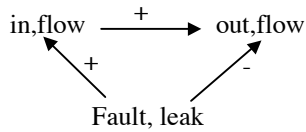
Equipment Model Builder identifies process variables with no propagation path through a unit model and provides the user with a list. This does not mean that the model is not complete if there are process variables with no propagation path. For example, in the case of the open tank there is no propagation path for flow from the tanks outports to its inports. It only means that the model *might* be not be complete. The list is intended to act as a memory aid for a user. This list of process variables with no propagation paths will help to ensure that the unit models behaviour is plausible.

## Correctness Techniques

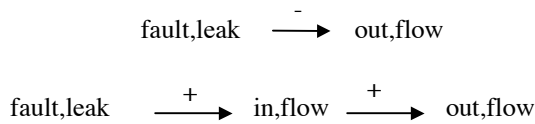
Correctness is checked for by identifying wrong information, looking for conflicting information within the model and preventing the entry of illegitimate attributes into the model. Deviations with no effect and deviations without causes may also occur because they contain wrong information.

Conflict may occur within the component model when

there is more than one possible path through the SDG and the paths have contradictory effects. The following SDG will be used to illustrate this:



From this SDG two propagation paths may be traced between the fault 'leak' and 'out,flow'. The two paths are:



For the first path the effect of a leak is a decrease of 'out,flow'. For the second path the effect is an increase of 'out,flow'. The qualitative analysis results in two contradictory paths with the first path having the correct influence.

In order to deal with ambiguities a heuristic that is commonly used is that when there is more than one acyclic path through the SDG the shortest path is used. A technique has been devised which allows the user to check that the shortest path within the component model leads to correct model behaviour. To avoid duplication of work this method utilises the QUEEN system (Chung 1993). A file of queries to test the effects of the shortest paths within the model is prepared. This file and the component model are given to the QUEEN system. The user checks the output from QUEEN to ensure that the model functions correctly.

Within the unit model, deviations may propagate to cause effects along four different types of path. The deviation may propagate:

1. from a boundary port to a boundary port;
2. from a fault to cause a consequence;
3. from a fault to a boundary port;
4. from a boundary port to cause a consequence.

Equipment Model Builder prepares an exhaustive list of queries to test for shortest paths between all the two node combinations which might occur within the model. The tool tests all possible paths. Not all of the paths tested for may exist. To provide a base for describing these queries a simple model is given below (in text). It is intended for illustration only.

```

frame(pipe isa unit,
  [ inports info [ in ],
    outports info [ out ],
    propLinks info [

%propagation

```

```

arc([in,pressure],+,[out,pressure]),
arc([out,pressure],+,[in,pressure]),
arc([in,temperature],+,[out,temperature]),
arc([in,flow],+,[out,flow]),
arc([out,flow],+,[in,flow]),

```

```

%faults
arc([fault,'partly blocked'],-,[out,flow]),
arc([fault,['leak into vacuum system',
vacuum]],+,[in,pressure]),

%consequences resulting from faults
arc([fault,'leak to environment'],+,[
consequence,['contaminate environment', toxic]],),
arc([fault,'leak to environment'],+,[consequence,
loss of material']),

%consequences resulting from deviations
arc([deviation,[morePressure,in]],+,[
consequence,'possible rupture'])
]
)

```

To verify type (1) paths a complete list of process variable deviations is prepared for every boundary port. Queries are prepared to test for shortest paths between each of the process variable deviations at a boundary port and each of the process variable deviations at the other boundary ports within the model. Examples of some of the paths queried for the pipe model shown are 'in,pressure' propagating to 'out,pressure', 'in,pressure' propagating to 'out,temperature' and 'out,pressure' propagating to 'in,pressure'. Queries are also prepared to test for shortest paths for deviations propagating from a boundary port back to the same port but a different process variable, e.g. 'in,flow' propagating to 'in,temperature'. Queries are not set up for deviations propagating from a boundary port back to the same variable at the same port, e.g. 'in,flow' propagating to 'in flow'.

To verify type (2) shortest paths Equipment Model Builder compiles a list of faults from the fault linked to deviation arcs within the model. Equipment Model Builder also creates a list of consequences contained in the deviation linked to consequence arcs within the model. Consequences contained in the fault linked to consequence arcs are not added to the list as the shortest paths propagating to these consequences are known. Queries are prepared to test for the shortest paths between each of the faults in the fault list and each of the consequences in the consequence list. The paths queried for the pipe model would be 'partly blocked' propagating to 'possible rupture' and 'leak into vacuum system' propagating to 'possible rupture'.

For type (3) paths queries are written to test for shortest paths from each of the faults in the fault list to each of

process variable deviations at each of the boundary ports. Examples of some of the paths queried for the pipe model are 'partly blocked' propagating to 'in,pressure', 'leak into vacuum system' propagating to 'in,pressure' and 'leak into vacuum system' propagating to 'out,temperature'.

To test type (4) paths queries are prepared to test for shortest paths between each of the process variable deviations at each of the boundary ports and each of the consequences in the consequence list. Examples of paths queried for the pipe model would be 'in,pressure' propagating to 'possible rupture', 'in,temperature' propagating to 'possible rupture' and 'out,pressure' propagating to 'possible rupture'.

As all possible shortest paths within the model are tested for, the user is able to see where paths do not exist as well as where they do. The omission of an arc may mean that a path does not exist where the user might expect to find one. QUEEN returns the shortest path and (where relevant) the effect of this path. The effect is not relevant for type (4) paths. The value of the deviations propagating into the unit's boundary ports in these paths is not known. Therefore it is not known whether these deviations will have an effect (i.e. cause a consequence) or not. QUEEN's output enables the user to check that both the shortest path and the effect are correct.

The technique for verifying the shortest path is intended to be the final verification procedure performed upon the model. Errors which could result in missing or incorrect shortest paths such as deviations with no effect and deviations without causes are detected prior to this test to reduce the number of problems found. The model at this stage should be complete and correct to the best of the user's knowledge. The user tests the unit model using the technique for verifying the shortest path. If any errors are found within the model the user will correct them and test the model again. A cycle of testing and correcting the model is carried out until the user is satisfied with it.

The layout of Equipment Model Builders front end interface prevents the user from entering illegitimate attributes. For example, the user is prevented from entering an arc containing a deviation which propagates through a port to influence itself at the same port, e.g. ([in,flow], +, [in,flow]).

## Conciseness Techniques

Equipment Model Builder checks for conciseness by testing for redundant information and preventing the addition of duplicated information to the model. Deviations with no effect and deviations without causes might be present because the information they contain is redundant. Equipment Model Builders user interface prevents the addition of duplicate arcs to the model.

## Case Studies

Two case studies are presented to assess whether the

verification techniques are effective. The application QUEEN (Chung, 1993) was employed to demonstrate the use of the models created by using Equipment Model Builder. For each test case models of the plant units were built and verified using Equipment Model Builder. These models plus a description of the plant topology were supplied to QUEEN.

## Plant Descriptions

The test cases are public domain examples of plant systems. The first case consists of the purification section of a plant producing benzene (Wells and Seagrave, 1976). The second case is an olefin dimerisation plant (Lawley, 1974).

**Benzene Purification System.** This purification system forms part of a plant producing benzene by the catalytic dehydroalkylation of toluene (Wells and Seagrave, 1976). The test case was restricted to the purification section of this plant in order to provide a relatively small example. The plant description used is given in figure 1. The distillation column (T101) separates the toluene and benzene components in its feed. The benzene is produced as the top product and the toluene as the bottom product. The top product is condensed by cooling water in exchanger E104 and then collected in reflux drum D103. The benzene is pumped by pump P101a from the reflux drum and is divided into a reflux stream and a product stream. The product stream is cooled by cooling water in exchanger E105 before going to storage.

Pump P101b is spare. All the valves in the system have open apertures. Figure 1 differs from the Wells and Seagrave plant in that there is no kick-back line from the P101 pumps and control facilities are omitted. The control facilities are omitted as Equipment Model Builder is currently unable to create unit models that have control structures.

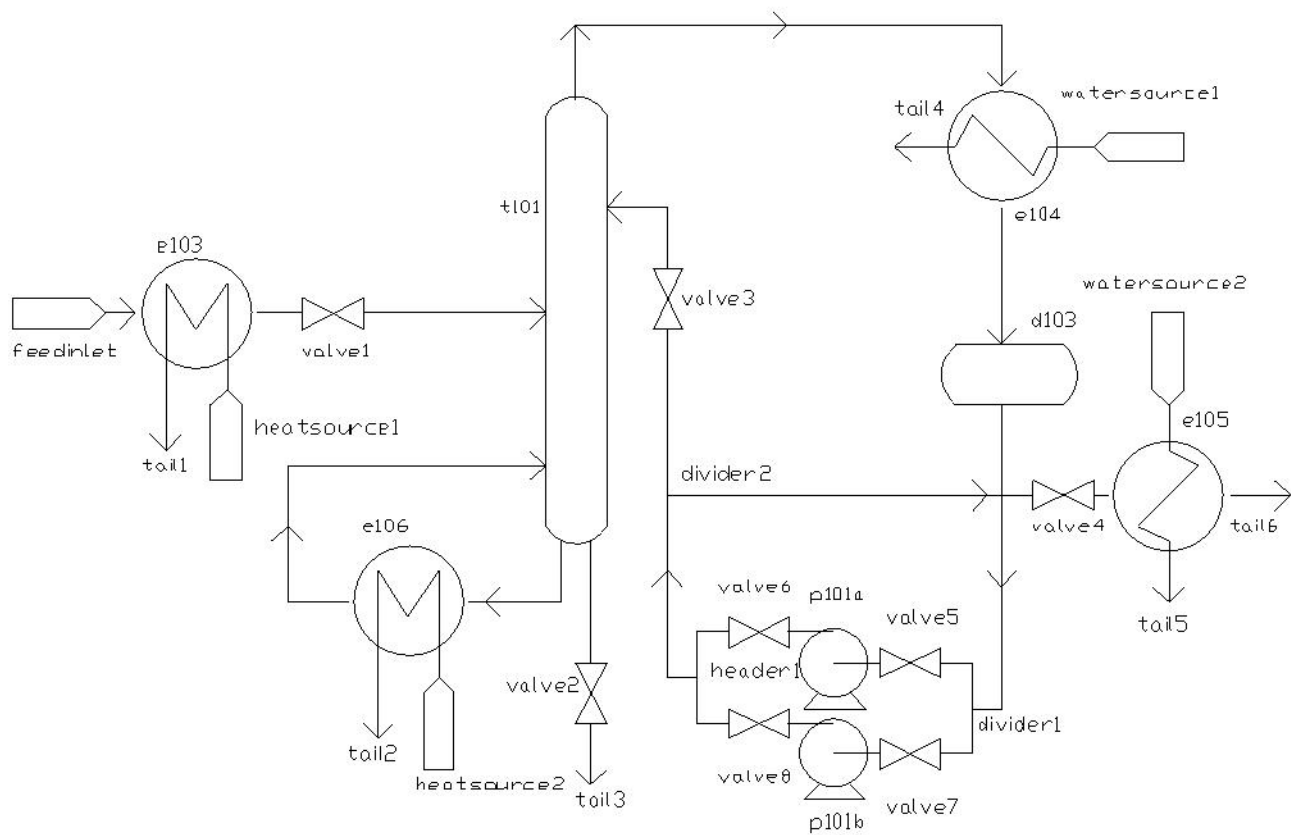


Figure 1. Benzene Purification System

**Olefin Dimerisation Plant.** The plant description of this test case is given in figure 2 and is based on figure 1 from Lawley (1974). Pumps J1a and J2a are working, pumps J1b and J2b are spare. Valves 5, 6, 9, 13 and 14 have closed apertures. All other valves in the plant have open

apertures. Notable differences between this plant description and Lawley's example are the omission of kickback lines from the J2 pumps, the pressure relief valve on the heat exchanger and of control facilities for the plant.

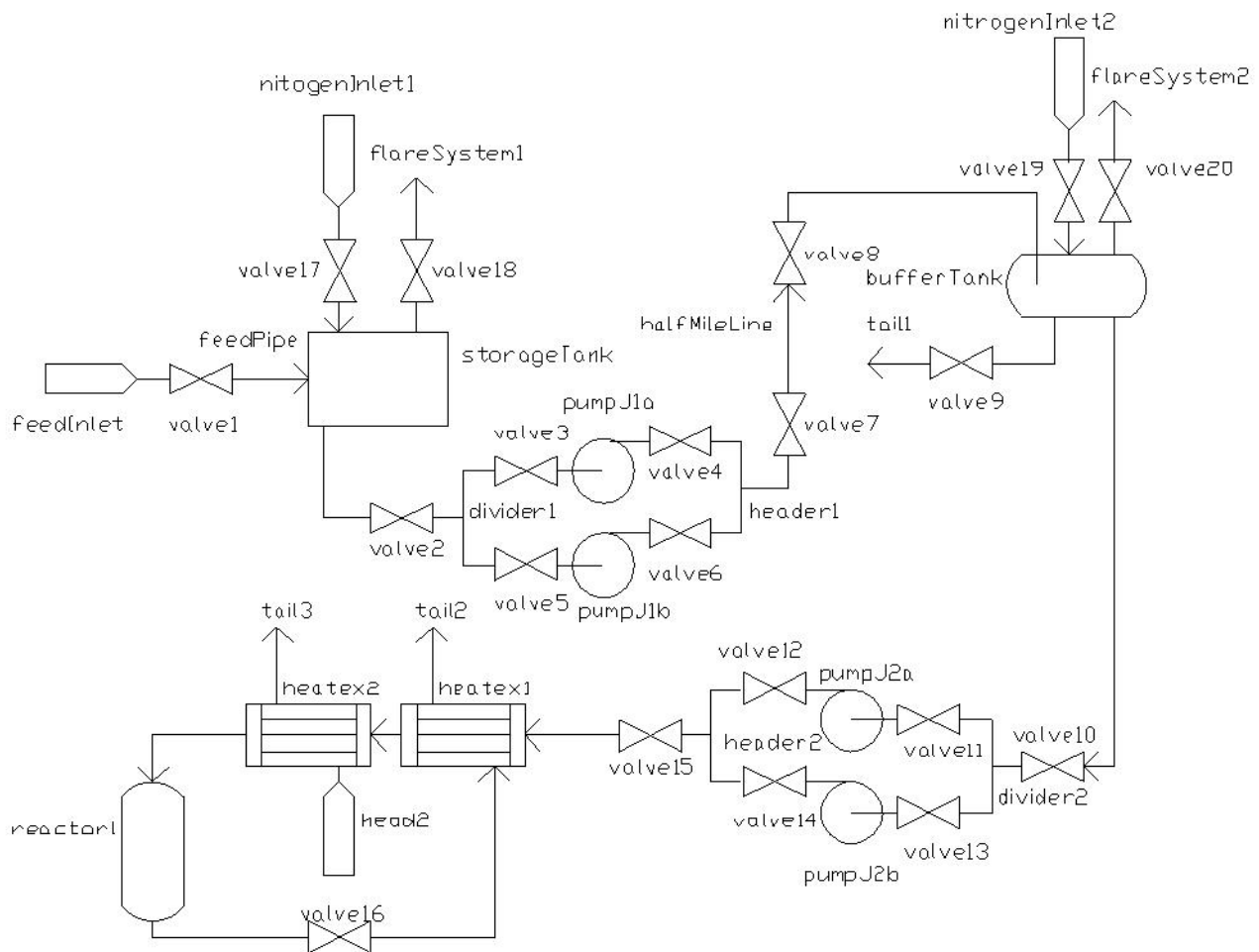


Figure 2. Olefin Dimerisation Plant

### Evaluation of the Verification Techniques

Verification of the models was found to be useful. The techniques found to be of greatest benefit were those used to check for deviations with no effect, deviations without causes and the technique for verifying the shortest path. The presence of deviations with no effect or deviations without causes provided a clear indication that the model created was faulty. It was found most useful to check for the presence of deviations with no effect and deviations without causes when the model was considered to be complete to check that it was not faulty.

Verifying the shortest path presented a different view of the model, thus allowing mistakes to be detected. This technique detected the following types of errors:

- simple mistakes;
- unexpected paths;
- missing arcs.

Simple mistakes caused paths to be present where they were not expected. An example of a simple mistake located in the models created for the case studies is:

'arc([in,level],+,[out,flow])', should be 'arc([in,flow],+,[out,flow])'.

This is an entry error caused by the user selecting the wrong variable from a menu.

Unexpected pathways occur when arcs the user has entered into the model interact in a way which the user did not expect. For example when creating a divider model for the case studies, assume the user has entered the following arcs:

'arc([out2,flow],+,[in,flow])'  
'arc([in,flow],+,[out1,flow])'

The divider model is instantiated as 'divider1' and 'divider2' in both the benzene purification system and olefin dimerisation plant. The arcs shown are both individually correct but lead to the path:

$$\text{out2,flow} \xrightarrow{+} \text{in,flow} \xrightarrow{+} \text{out1,flow}$$

The effect of this path would be for an increase in 'out2,

flow' to increase 'out1,flow'. This may be not intended to occur within the model.

Missing arcs may be detected when expected propagation paths in the model are not found or when unexpected propagation paths are found. For example, if 'arc([out2,flow],-[in,flow])' were missing from the divider model the unexpected propagation path shown above might result.

A limitation with the technique for verifying the shortest path is that it is exhaustive. All possible paths within the model are tested for. This results in a large numbers of queries generated and a large amount of information for checking. Possible ways of simplifying the results would be:

- to list positive output (where paths exist) in a separate file to negative output;
- to allow the user a choice of which results to view.

The user may not wish to view all of the results generated. The user may only wish to look at paths which contain more than two arcs or those containing a certain process variable deviation. Looking at paths containing two or more arcs would detect errors caused by the unforeseen interaction of arcs, which the user may feel to be the most likely source of error.

The user may feel it is only necessary to view paths containing a certain process variable. For example, if a heatexchanger were being modeled the user might decide that only the shortest paths containing the process variable deviation temperature were of interest. The other paths might be assumed to be correct. If a blanketedvessel model were being created by extending a closedvessel model the user might wish to limit the results to those shortest paths containing pressure. These paths would be present as a result of new arcs added when the closed vessel model was extended. Other shortest paths within the blanketedvessel model would already have been verified as the arcs containing these paths would have been copied from the closedvessel model. The closedvessel model would have been verified when it was created. The user may also wish to condense the results viewed in other ways. Providing a choice would allow the user to restrict the output to those results of interest.

## Conclusions

Verification is necessary to detect modelling errors which may give rise to wrong results when the models are utilised. The aim of verification is to ensure as far as possible that a model is complete, correct and consistent. In order for a model to be consistent it must be correct and concise.

A series of verification techniques for signed directed graph models has been described. Case studies have demonstrated these techniques to be of use. However, some modelling errors may still remain as a user may be unaware what information is missing or incorrect.

## References

- Catino, C.A.; Grantham S.D.; and Ungar L.H. 1991. Automatic Generation of Qualitative Models of Process *Computers and Chemical Engineering*, 15: (8) 583-599.
- Chung, P.W.H. 1993. Qualitative Analysis of Process Plant Behaviour, *The Proceedings of the Sixth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*: 277-283.
- Gupta, U.M. 1993. Validation and Verification of Knowledge-Based Systems: A Survey. *Journal of Applied Intelligence*, 3: 343-363.
- Iri, M.; Aoki, K.; O'Shima, E.; and Matsuyama, H. 1979. Algorithm for Diagnosis of System Failures in the Chemical Process. *Computers and Chemical Engineering*, 3 (1-4): 489-49.
- Larkin, F. D.; Rushton, A. G.; Chung, P. W. H.; Lees, F. P., McCoy, S. A. and Wakeman, S. J. 1997. Computer-Aided Hazard Identification: Methodology and System Architecture, *ICHEME Symposium Series Hazards XIII Process Safety - The Future*, 141: 337-348, ICHEME, Rugby, UK.
- Lawley, H.G. 1974. Operability Studies and Hazard Analysis. *Chemical Engineering Progress*, 70 (4): 45-56.
- Oyeleye, O. O.; and Kramer, M.A. 1988. Qualitative Simulation of Chemical Process Systems: Steady State Analysis, *AIChE Journal*, 34: 1441-1454.
- Palmer, C.; and Chung, P.W.H. 1997. Constructing Qualitative Models, *ICHEME Jubilee Research Event*, 2: 725-728, ICHEME, Rugby, U.K.
- Palmer, C.; and Chung, P.W.H. 1998. Eliminating Ambiguities in Qualitative Models, *Computers and Chemical Engineering*, (Suppl) 22: S843-S846.
- Schut, C.; and Bredeweg, B. 1994. Supporting Qualitative Model Specification, *Proceedings of the Second International Conference on Intelligent Systems Engineering*: 37-42.
- Wells, G.L. and Seagrave, C.J. 1976. Flowsheeting for Safety, a guide on safety measures to consider during the design of chemical plant, ICHEME.
- Wilcox, N.A.; and Himmelblau, D.M. 1994. The Possible Cause and Effect Graphs (PCEG) Model for Fault Diagnosis -I. Methodology, *Computers and Chemical Engineering*, 18: 103-116.
- Vaidhyanathan, R.; and Ventkatasubramanian, V. 1995. Digraph-Based Models for Automated HAZOP Analysis, *Reliability Engineering and System Safety*, 50, 33-49.