COMPUTER-AIDED HAZOP OF BATCH PROCESSES

Paul W.H. Chung and Steve A. McCoy
Department of Computer Science, Loughborough University, Loughborough, Leicestershire LE11 3TU; UK; email: p.w.h.chung@lboro.ac.uk

HAZOP is a well-respected technique for identifying potential hazards in process plants. However, the application of the technique is very repetitive and time consuming. One way to overcome this bottleneck is to develop automated hazard identification systems that emulate the HAZOP technique. Much of the research on automated hazard identification so far has concentrated on continuous plants by considering the causes and consequences of deviations from steady state. This paper focuses primarily on batch processes where a batch plant moves through a number of different stages during operation. It investigates the use of qualitative simulation and knowledge-based techniques to analyse the effect of following a sequence of operating instructions on a plant. A prototype system called CHECKOP is described. The proposed ideas will be illustrated through an example. Further ideas for extending the computational framework will also be discussed.

INTRODUCTION

HAZOP studies are widely used for identifying hazard and operability problems during plant design. The technique is described in a number of books (for example Kletz, 1999). However, HAZOP studies are time consuming, labour intensive and expensive. Therefore, major research projects have been carried out to develop tools to automate the HAZOP technique. Some advances have been made in the area of automated hazard identification of continuous plants by considering the causes and consequences of deviations from steady state. The computing technique used is generally based on the idea of fault propagation using signed-directed graph (Chung, 1993; Jefferson et al., 1995; Vaidhyanathan and Venkatasubramanian, 1995). While a lot of progress has been made in this area (McCoy et al., 1999a, 1999b and 1999c; 2000a and 2000b), very little work has been done in automated hazard identification of batch plants. The signed-directed graph technique is inappropriate for batch processes as a batch plant moves through a number of different stages during operation (McCoy et al., 2003). Therefore, a new approach has to be sought.

The rest of this paper describes a research project that considers the use of qualitative simulation and knowledge-based techniques to analyse the effect following a sequence of operating instructions has on a plant. The consequences of missing out instructions or carrying them out too early or too late are also analysed. A research prototype system called CHECKOP has been designed and implemented to test these ideas. A description of an earlier version can also be found in Chung et al. (2003).

CHECKOP SYSTEM OVERVIEW

The CHECKOP system consists of three main components, takes a number of files as input and generates a report file as output as shown in Figure 1.

The three main components of the system are the *Parser*, the *Deviation Generator* and the *Simulation Engine*. The *Parser* reads the input files prepared by the user and converts the information into an internal form for processing by the other two components. The information provided by the user is specific to the plant that is required to be HAZOPed. One of the files gives details about the items of equipment in the plant, their connectivities and their current states. The other file contains a set of operating instructions to be applied to the plant to bring the plant from its current state to its goal state, while also achieving the production of a batch of product!

The *Deviation Generator* systematically applies the deviation guidewords — no, early and late — to the operating procedure so that the *Simulation Engine* can infer what will be the consequence if a certain instruction in the procedure is not executed, or the instruction is carried out too early or too late. Having gone through all the deviations, the *Simulation Engine* will produce a report file providing warnings against any undesirable situations that may result from the deviations. To carry out the simulation the *Simulation Engine* requires the Action Model Library which provides information about actions that can be performed on different pieces of equipment and the effects of those actions.

PLANT DESCRIPTION

An object-oriented approach is used to describe the plant. Consider the batch plant as shown in Figure 2; each item in the plant is declared in the plant description file. For

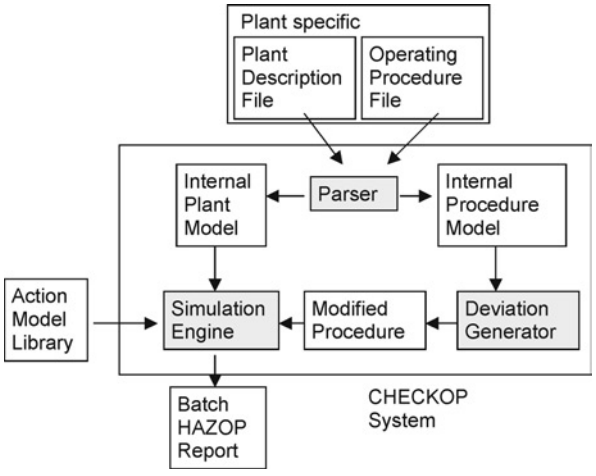

Figure 1. The components of CHECKOP

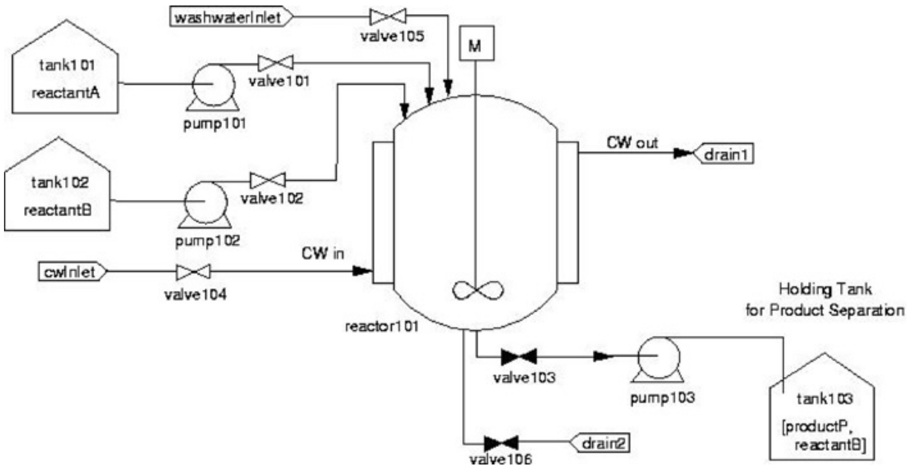

Figure 2. A simple batch plant

each item at least the following basic information is given:

- The type of unit it belongs to;
- Which other plant items it is connected to.

Other appropriate information related to a plant item will also be stored with that plant item. Table 1 provides some example descriptions of the plant items found in Figure 2.

OPERATING PROCEDURE DESCRIPTION

In order for the CHECKOP system to analyse an operating procedure, the instructions have to be written following the templates. In general, instructions that are written in natural language style are difficult for the computer to understand and their meaning may also be ambiguous. Therefore, to avoid natural language processing, an operating procedure written as input to CHECKOP strictly follows the templates below:

Template 1: *Item Action*
Example: valve101 open

Template 2: *Item Action* until *Condition*
Example: mixer on until elapsed-time 20 minutes

Template 3: *Item1 Action Item2 Filler-word Fluid* until *Condition*
Example: reactor101 fill-from tank101 with reactantA until volume 30 percent

Table 1. Explanation of Plant Description

Formal plant item declaration	Explanation
instance(tank101 is a tank,	Tank101 is a tank
[	
content info [reactantA],	The content of the tank is reactantA
outports info [out is [pump101,in]]	The outlet of the tank is connected to the inlet of pump101
]).	
instance(pump101 is a pump,	Pump101 is a pump
[	
status is offline,	The status of the pump is off-line
outports info [out is [valve101,in]]	The outlet of the pump is connected to the inlet of valve101
]).	
instance(valve101 is a valve,	Valve101 is a valve
[	
status is closed,	The status of the valve is closed
outports info [out is [reactor101, in2]]	The outlet of the valve is connected to inlet 2 of reactor101
]).	
instance(reactor101 is a stirred_tank_reactor,	Reactor101 is a stirred-tank-reactor
[	
outports info [out1 is [valve103,in], out2 is [valve106,in]],	The outlet 1 of the reactor is connected to the inlet of valve 101 and outlet 2 is connected to valve 106
heatSink info [hout is [jacket101,hin]],	The heat of the reactor is transferred to jacket 101
reaction info [reaction_ab_p]]).	The intended reaction is called reaction_ab_p

Given the plant shown in Figure 2, the instructions for charging reactor101 with reactantA can be expressed as:

```
valve101 open
pump101 start
reactor101 fill-from tank101 with reactantA until volume 30
  percent
pump101 stop
valve101 close
```

The file containing the operating instructions for operating the plant is read in by CHECKOP and translated into its internal form.

THE ACTION MODEL LIBRARY

Associated with each plant item type there is an action model in the *Action Model Library*. The model specifies the operations that can be carried out on that type of plant item. For each action the pre-conditions that must be true before the action and the post-conditions after the action are stated.

For example the actions for a valve can be *open* or *close*. In its general form, there is no pre-condition for opening or closing a valve. However, the post-condition for opening a valve is that a flow path exists between the upstream unit and the downstream unit. The post-condition for closing a valve is that the flow path between the upstream unit and the downstream unit no longer exists.

The actions for a pump can be *start* or *stop*. To start a pump, the pre-conditions are that there must be a flow path between the source of a fluid and the pump and there must be a flow path between the pump and the sink. If the pre-conditions are not met then *start* operation will generate a warning message. The post-condition of starting a pump is that there is a flow between the source and the sink. On the other hand, there is no pre-condition for stopping a pump and the post-condition of stopping a pump is that there is no flow between the source and the sink.

THE DEVIATION GENERATOR

The Deviation Generator applies the guide words *no*, *early* and *late* systematically to the operating instructions to generate different versions of the operating procedure. This allows CHECKOP to explore the consequences of different scenarios that could result from operator human errors.

By applying the guideword *no* to the following example procedure:

```
(1) valve101 open
(2) pump101 start
(3) reactor101 fill-from tank101 with reactantA until volume
    30 percent
(4) pump101 stop
(5) valve101 close
```

the Deviation Generator will remove systematically one instruction at a time from the procedure, which will result in five different procedures. Each representing an error of omission, i.e. an operator failed, or forgot, to carry out a specified instruction.

For example, procedure with instruction 1 omitted:

```
(2) pump101 start
(3) reactor101 fill-from tank101 with reactantA until volume
    30 percent
(4) pump101 stop
(5) valve101 close
```

Procedure with instruction 2 omitted:

```
(1) valve101 open
(3) reactor101 fill-from tank101 with reactantA until volume
    30 percent
(4) pump101 stop
(5) valve101 close
```

When the guide word *early* is applied to the procedure, instructions are moved earlier in the procedure. For example, moving the instruction "reactor101 fill-from tank101 with reactantA until volume 30 percent" two steps forward will result in the procedure:

```
(3) reactor101 fill-from tank101 with reactantA until volume
    30 percent
(1) valve101 open
(2) pump101 start
(4) pump101 stop
(5) valve101 close
```

All the different procedures generating by the Deviation Generator are passed to the Simulation Engine for analysis to identify operability problems and potential hazardous situations.

SIMULATION ENGINE

The heart of the CHECKOP system is the simulation engine. Given an operating procedure, it applies the instructions one at a time and simulates its effect by changing the state of the plant. Therefore, the plant moves from one state to another until all the instructions are completed. However, the execution of a procedure may not always reach its end. This is because when the simulation engine detects an operability problem or hazardous situation it will report to the user.

For example, an analysis of the example procedure with instruction one missing will result in the following warning: There is no flow path between tank101 and reactor101 for filling.

The simulation engine will work systematically through all the procedures generated by the deviation generator.

SYSTEM IMPLEMENTATIONS AND SAMPLE RUNS

The system has gone through two iterations of design and implementation. The initial prototype of CHECKOP was written using the knowledge-based system toolkit CLIPS (Donnel and Riley, 1994) to prove the concepts. Not all the components described in the previous sections were implemented. The following example illustrates the capability of the early prototype. Consider a simple procedure for filling a kettle written in

CLIPS syntax:

```
1. (op [kettle] move-under [kitchen-tap])
2. (op [kettle] open-lid)
3. (op [kitchen-tap] turn-on)
4. (op [kettle] fill-from [kitchen-tap] with water until
   volume 50 percent)
5. (op [kitchen-tap] turn-off)
6. (op [kettle] close-lid)
```

Different versions of the procedure are created manually by applying the guidewords *no* and *other*. The effects of following the modified procedures are determined by CHECKOP.

Example 1: with instruction 1 removed, CHECKOP generates the warning:

```
Cannot fill [kettle] because it is not under the [kitchen-
tap] for filling.
```

Example 2: with instruction 2 removed, CHECKOP generates the warning:

```
Cannot fill [kettle] because the lid is closed.
```

Example 3: with instruction 3 removed, CHECKOP generates the warning:

```
Cannot fill [kettle] because there is no flow from [kitchen-tap].
```

Example 4: with instruction 1 replaced with

```
(operation [kettle] move-under [softdrink-tap])
```

CHECKOP generates the warning:

```
Cannot fill [kettle] because it is not under the [kitchen-
tap] for filling.
```

Example 5: with `[kitchen-tap]` replaced by `[softdrink-tap]` in all the instructions, CHECKOP generates the warning:

```
Cannot fill [kettle] because the content of [softdrink-tap]
is not water.
```

CHECKOP is being re-designed and re-implemented in C++ to run under Windows. The Deviation Generator that was not implemented in the CLIPS prototype is now in the C++ version. However the Action Model Library is still very limited. Furthermore, CHECKOP also needs to be extended with a new module to handle consequence modelling so that the result of the analysis is not limited to the immediate effect of the operating problems but will be able to identify the hazards that follow.

CONCLUSIONS AND FUTURE WORK

Significant progress has been made in developing systems to automate hazard identification. The technology for continuous operations has reached maturity for commercialisation. Much work still has to be done for batch operation, but promising progress has been made.

The current version of CHECKOP has demonstrated proof of concept for modelling operating procedures and their effect on a simple batch process. However, the Action Model Library is still very limited and requires further development work, to cover a wider range of actions; additionally, some of the existing action models need enhancing, to ensure that their pre-conditions and post-conditions are as accurate as possible. It may also be possible to develop the operations used to group actions too, so that standard operations models can be defined and reused many times, in the same way that individual actions are defined from a model.

In parallel with the Action Model Library development, a wider range of deviation guidewords must be considered, as automatically applied to actions in the operating instructions. So far, "No Action" and "Early/Late Action" have been considered and modelled. The next step is to fully consider "Shorter/Longer Action", where the action is performed for a longer or shorter time than intended. This is particularly applicable for those actions which have an "until" clause (e.g. what happens if the reactor is only filled to 10% instead of 30%?). Variants of the "Other Action" guide word should also be considered. This would include considering what happens if the action is performed on the wrong piece of equipment (e.g. what if the wrong pump is switched off?). Care needs to be taken to ensure that the most sensible and likely deviations are created for a given guide word, as the potential number of variations for even a single action step is huge.

The earliest version of CHECKOP was developed in CLIPS, and the program is now being re-designed and re-implemented in C++ to run under Windows. This has the benefit of faster performance and is also a more commonly known language among software developers, so that future development should be facilitated. One other goal is to integrate the CHECKOP tool with the already written Hazid system (Chung et al., 2004), for Hazop analysis of continuous plants. The integration of hazard identification tools for both continuous and batch operations will provide a complete system for HAZOP that is able to handle continuous operation and the non-steady state operation of the same plant, such as start up and shut down. This also means that the batch Hazop simulator will have access to the full process behaviour models provided by Hazid. Since Hazid is written in C++, it is natural to move to the same language for CHECKOP.

Within CHECKOP, further development will also be needed to model phenomena which are not so far covered very well, such as process fluids and their interactions and reactions (whether intended or unwanted). This work will feed into more accurate modelling of the consequences of mal-operation in the batch plant, in terms of chains of events and/or hazards — a form of consequence modelling. The objective in doing this type of modelling is to provide an even richer form of output as a result of the batch Hazop emulation performed by CHECKOP.

ACKNOWLEDGEMENT

The work described in this paper is supported by a grant from the Engineering and Physical Sciences Research Council (Grant number: GR/R37531).

REFERENCES

Chung, P.W.H. (1993) Qualitative Analysis of Process Plant Behaviour, Proceedings of the Sixth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Chung, P.W.H., Lovegrove, G. and Ali, M. (eds), Gordon and Breach Science Publishers, Edinburgh, June 1993, pp. 277–283.

Chung, P.W.H., Wen, Q., Connolly, J.H., Busby, J.S. and McCoy, S.A. (2003) Knowledge-based Support for the Authoring and Checking of Operating Procedures, Proceedings of the 16th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Chung, P.W.H., Hinde, C. and M. Ali (eds), Springer, Berlin, Loughborough, June 2003, pp. 264–270.

Chung, P.W.H., Lam, W., Lee, R., Madden, J., McCoy, S. and Wilson, T. (2004) Integration of Hazard Identification Software with Process Design Systems, Proceedings of APCChE 2004, Japan. (To appear).

Donnel, D. and Riley, G. (1994), CLIPS Reference Manual Version 6.0, Software Technology Branch, Lyndon B. Johnson Space Center, Houston, Texas.

Jefferson, M., Chung, P.W.H. and Rushton, A.G. (1995) Automated hazard identification by emulation of hazard and operability studies. Procedings of 8th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Melbourne, pp 765–770. Gordon and Breach Publishers.

Kletz, T.A. (1999) Hazop and Hazan: Identifying and Assessing Process Industry Hazards (4th Edition), IChemE.

McCoy, S.A., Wakeman, S.J., Larkin, F.D., Jefferson, M., Chung, P.W., Rushton, A.G., Lees, F.P. and Heino, P.M. (1999a) HAZID, A Computer Aid for Hazard Identification 1. The STOPHAZ Package and the HAZID Code: An Overview, the Issues and the Structure, Transactions of the Institution of Chemical Engineers, Part B, 77, pp. 317–327.

McCoy, S.A., Wakeman, S.J., Larkin, F.D., Chung, P.W., Rushton, A.G. and Lees, F.P. (1999b) HAZID, A Computer Aid for Hazard Identification 2. Unit Model System, Transactions of the Institution of Chemical Engineers, Part B, 77, pp. 328–334.

McCoy, S.A., Wakeman, S.J., Larkin, F.D., Chung, P.W., Rushton, A.G., Lees, F.P. and Heino, P.M. (1999c) HAZID, A Computer Aid for Hazard Identification 3. The Fluid Model and Consequence Evaluation Systems, Transactions of the Institution of Chemical Engineers, Part B, 77, pp. 335–353.

McCoy, S.A., Wakeman, S.J., Larkin, F.D., Chung, P.W., Rushton, A.G. and Lees, F.P. (2000a) HAZID, A Computer Aid for Hazard Identification 4. Learning Set, Main Study System, Output Quality and Validation Trials, Transactions of the Institution of Chemical Engineers, Part B, 78, pp. 91–119.

McCoy, S.A., Wakeman, S.J., Larkin, F.D., Chung, P.W., Rushton, A.G. and Lees, F.P. (2000b) HAZID, A Computer Aid for Hazard Identification 5. Future Development Topics and Conclusions, Transactions of the Institution of Chemical Engineers, Part B, 78, pp. 120–142.

McCoy, S.A., Zhou, D. and Chung, P.W.H. (2003), State-Based Modelling in Hazard Identification, Proceedings of 16th International Conference on Industrial and Engineering

Applications of Artificial Intelligence and Expert Systems, Chung, P.W.H., Hinde, C.J. and Ali, M. (eds), Springer-Verlag, Loughborough University, Loughborough, UK, June 2003, pp. 244–253.

Vaidhyanathan, R. and Venkatasubramanian, V. (1995) Digraph-based models for automated HAZOP analysis, Reliab Eng System Safety, 50:33.