Formal development of remote interfaces for large-scale real-time systems

2004 IEEE International Conference on Systems, Man and Cybernetics

Formal Development of Remote Interfaces for Large-Scale Real-Time Systems*

Walter Hussak, Shuang H. Yang
Department of Computer Science
Loughborough University
Loughborough, United Kingdom
{W.Hussak, S.H.Yang}@lboro.ac.uk

Abstract – *The design of web-based user interfaces is of primary importance for achieving successful operation of Internet-based monitoring and control systems. Operators need to be able to act promptly on changing situations requiring remote actions to process plants. A formal development process is proposed to determine the minimum amount of information that needs to be presented at interfaces. The first stage of the process is a specification of states of components that require operator actions. The main stage of the process uses model checking to generate interfaces with a minimal amount of information sufficient for the operator to perform all required actions. As well as improving the efficiency of operators, simpler interfaces allow for greater concurrency in the implementation of the remote operation of the process plant.*

Keywords: Remote interfaces, control systems, model checking, concurrency.

1 Introduction

The Internet offers a great platform for establishing truly distributed large-scale systems, such as Internet-based monitoring and control systems for process plants. One of the challenges in the design of Internet-based monitoring and control systems is to produce suitable web-based user interfaces enabling operators to appreciate quickly what is happening in the process plants located at a remote site [7]. It should be borne in mind that requests for large amounts of information in the interface increase the transmission load over the Internet and also limit scope for concurrency between interfaces requiring a consistent view of common data. As a result, the speed of communication is slowed down. Furthermore, overloading information in the interface may obscure how the operator should act on the information presented and thus slow down the response of the operator. The objective of this paper is to give a formal development process for producing remote interfaces which provide the optimal amount of information. The general process is given in Section 2 and a case study, where remote interfaces for a real-life process plant are developed, is described in Section 3. The conclusions are in Section 4.

2 General development process

A three-stage development process is proposed for each interface:

1. Give an initial formal specification of the interface comprising information on a set of components required by the remote operator in order to operate the particular part of the plant, the states that have to be attained by these components before the operator performs each action, and the changes each action makes to the state of components. At this stage it may not be clear where specified required information has actually been overloaded and can be reduced by taking into account relationships between different components in the plant.
2. Generate by formal verification, alternative candidate sets of components, from which information on the enabling states of the components in stage 1 can be inferred, which are optimal with respect to the least amount of information being presented at the interface.
3. The final interface results from choosing a candidate set of components from stage 2, based on any criteria, additional to minimizing the amount of information presented, that are not within the scope of this paper.

2.1 Initial interface specification

In Internet-based monitoring and control systems a large-scale processing plant might be controlled remotely by a number of the operators located in different parts of the world. Each of them would be given responsibilities for the operation of different parts of the plant. The basic problem when designing the interface for the remote user is to decide what information to present to the operator. The initial specification addresses three aspects:

(i) Components of the plant that an operator is empowered to control, i.e. change their state;

* 0-7803-8566-7/04/$20.00 © 2004 IEEE.

(ii) Components of the plant whose state influences the actions of the operator on the components in (i).
(iii) The states of the components in (ii) that may cause the operator to act on components in (i).
To specify these, notation from the theory of database transactions is adopted. The set of all the components in the plant will be denoted by X. If the operator i has the power to change the state of components Z1,..., Zq based on knowledge of the state of components Y1,..., Yp, where Y1,..., Yp, Z1,..., Zq belong to X, then the interface for the operator i, which corresponds to (i) and (ii) above, is specified as:

Ri[Y1,...,Yp], Wi[Z1,..., Zq]

This is analogous to two-step database transactions [5] in that operator i updates or 'writes' to variables Z1,..., Zq (denoted Wi[Z1,..., Zq]) based on the values observed or 'read' for Y1,...,Yp (denoted as Ri[Y1,...Yp]). It is assumed that {Z1,..., Zq} is a subset of {Y1,..., Yp}, i.e. an operator needs to know the state of components prior to any change. The set of subsets of {Y1,...,Yp} containing {Z1,...Zq} is denoted by Y. The states of the components Y1,...,Yp that cause the operator to act, corresponding to stage (iii) above, are presented as a finite set of sets of propositions:

U = {{Y1=U1,..., Yp=Up} : Y1 in state U1,..., Yp in state Up causes the operator to act}

There is an assumption here that there are finitely many distinct situations that cause the operator to act on Z1,..., Zq. This may require negations of propositions such as !Y1=U1, in the sets in U, which we have not mentioned at this stage for simplicity of exposition.

2.2 Generation of candidate interfaces

As it is the 'read set' Y1,..., Yp that gives the information presented at the interface, it is this set that is to be minimized. The general algorithm, below, for obtaining the list of minimal read sets `rsets`, loops through all possible subsets {X1,..., Xs} of {Y1,...,Yp} starting with s=1 (first and second `for` loops) to see if every enabling condition {Y1=U1,..., Yp=Up} for an action can be inferred from its subset {X1=V1,..., Xs=Vs} (third `for` loop). This involves executing the model checker NuSMV [1] to determine if the conjunction Y1=U1 &...&Yp=Up is always (AG) equivalent to X1=V1 &...&Xs=Vs in the operation of the plant. In the code NuSMV_call is therefore defined to be:

```
NuSMV(system, AG(Y1=U1&...&Yp=Up <->
                 X1=V1&...&Xs=Vs))
```

Where `system` is the logical behaviour of the plant. At the end, `rsets` will contain subsets {X1,..., Xs} of candidate read sets for a particular minimum cardinality s.

The full algorithm is as follows:

```
procedure generate_read_sets
  rsets := <>;
  min_intrfce_achieved := false;
  for s=1 to p do
    if min_intrfce_achieved=false then
      for {X1,..., Xs} in Y do
        matched:=true;
        for {Y1=U1,..., Yp=Up} in U do
          if matched then
            matched = NuSMV_call
          end if
        end for;
        if matched then
        rsets=append({X1,..., Xs},rsets);
        min_intrfce_achieved = true
        end if
      end for
    end if
  end for
end procedure
```

Although, the algorithm above does not introduce any new state explosion problem with the model checker NuSMV, optimizations to the algorithm are suggested in the case study in the next section to prevent excessively many calls to NuSMV.

3 Case study

The case study is a scaled-up version of the transferring system for a penicillin process given in [6]. The system is made up of two dimethyl acetamide (DMA) tanks, two reactors, two external pumps (mA, mB) for each tank, two Ellis locks (VA, VB) for each tank, and two control valves (vA, vB) for each reactor, as shown in Figure 1.

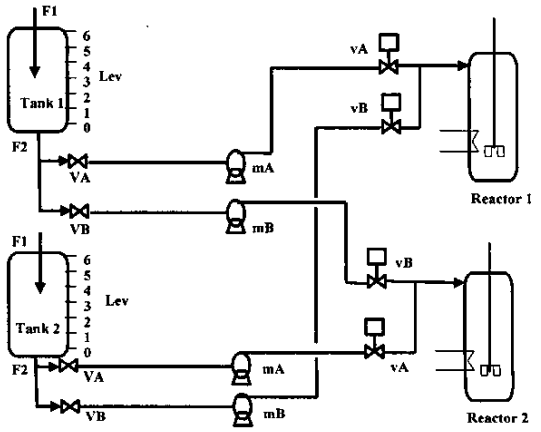

Figure 1. A penicillin process

The DMA tank holds the DMA solvent until is ready to be transferred to the reactors at the start of a new batch process. For each DMA tank, we assume that if both the inlet flow F1 and outlet flow F2 are open the inlet flow rate is greater than the outlet flow rate. Discrete integer values (0 to 6) have been chosen for the liquid level `Lev`.

3.1 Remote operator interface specification

Operator a is responsible for the transfer lines from tank1 to reactor1, and tank2 to reactor2, and operator b is responsible for the transfer lines from tank1 to reactor2 and tank2 to reactor1. Operator a can open tank1.VA, tank1.mA and tank1.vA if they are closed and tank1.VB, tank1.mB and tank1.vB are closed. Operator a can also open tank2.VA, tank2.mA and tank2.vA if they are closed and tank2.VB, tank2.mB and tank2.vB are closed. Operator b is defined to control VB, mB, and vB in a similar fashion. The initial specifications for operator a, for the aspects (i), (ii) and (iii) of 2.1, are:

```
Ra[tank1.VA,tank1.mA,tank1.vA,
   tank1.VB,tank1.mB,tank1.vB,
   tank2.VA,tank2.mA,tank2.vA,
   tank2.VB,tank2.mB,tank2.vB]
Wa[tank1.VA,tank1.mA,tank1.vA,
   tank2.VA,tank2.mA,tank2.vA]
Ua={{tank1.VA=U1,tank1.mA=U2,
   tank1.vA=U3,tank1.VB=U4,
   tank1.mB=U5,tank1.vB=U6,
   tank2.VA=U7,tank2.mA=U8,
   tank2.vA=U9,tank2.VB=U10,
   tank2.mB=U11,tank2.vB=U12} :
   U1=off&U2=off&U3=off&
   U4=off&U5=off&U6=off |
   U7=off&U8=off&U9=off&
   U10=off&U11=off&U12=off}
```

Here, Ua has been defined by set comprehension, where `:` is read as 'such that' and `|` is read as 'or'.

3.2 System behavior specification

The behavior of the overall plant is specified as a finite state machine in the input language of NuSMV shown as follows, in which the following keywords appear:

- MODULE: indicates the file as main or subroutine.
- VAR: define types of variables as boolean or sets of symbols.
- DEFINE: in order to make descriptions more concise, a symbol can be associated with a commonly used expression.
- SPEC: write specifications in CTL (computation tree logic).
- ASSIGN: define transition relations among variables.
- init: define initial conditions of the variables.
- next: define a relationship between values of variables in a particular state and its successor state.

```
1  MODULE main
2
3  VAR
4    oa: boolean; ob: boolean;
5    tank1: tank_subsystem(oa,ob);
6    tank2: tank_subsystem(oa,ob);
7
8  ASSIGN
9    init(oa):=0; init(ob):=0;
10
11 SPEC
12 AG(!tank1.VA & !tank1.mA & !tank1.vA
13 & !tank1.VB & !tank1.mB & !tank1.vB
14 <-> !tank1.VA &!tank1.mA &!tank1.vA
15 & !tank1.VB & !tank1.vB)
16
17 MODULE tank_subsystem(oA,oB)
18
19 VAR
20   Lev: {0,1,2,3,4,5,6};
21   F1: boolean; F2: boolean;
22   VA: boolean; VB: boolean;
23   mA: boolean; mB: boolean;
24   vA: boolean; vB: boolean;
25
26 DEFINE
27   higher:=case Lev<6:Lev+1;
28    Lev=6:6 esac;
29   lower:=case Lev>0:Lev-1;0:0;esac;
30   LL:= Lev<3;
31   LH:= Lev>4;
32
33 ASSIGN
34   init(Lev):=3;
35   next(Lev):=case F1&!F2:higher;
36    !F1&F2:lower;F1&F2:higher;
37     F1&F2:higher;1:Lev; esac;
38   init(F1):=0;
39   next(F1):=case LH:0;1:{0,1};esac;
40   init(F2):=0;
41   next(F2):=case LL:0;1:{0,1};esac;
42   init(VA):=0;
43   next(VA):=case LL:0;oa&!ob&!VA
44    &!mA&!vA&!VB&!mB&!vB:1;1:VA;esac;
45   init(VB):=0;
46   next(VB):=case LL:0;ob&!oa&!VB
47    &!mB&!vB&!VA&!mA&!vA:1;1:VB;esac;
48   init(vA):=0;
49   next(vA):=case vA&!mA:0;oa&!ob&!VA
50    &!mA&!vA&!VB&!mB&!vB:1;1:vA;esac;
51   init(vB):=0;
52   next(vB):=case vB&!mB:0;ob&!oa&!VB
53    &!mB&!vB&!VA&!mA&!vA:1;1:vB;esac;
54   init(mA):=0;
55   next(mA):=case mA&!VA:0;oa&!ob&!VA
56    &!mA&!vA&!VB&!mB&!vB:1;1:{mA,0};
57     esac;
58   init(mB):=0;
59   next(mB):=case mB&!VB:0;ob&!oa&!VB
```

```
60      &!mB&!vB&!VA&!mA&!vA:1;1:{mB,0};
61    esac;
```

In the finite state machine for the penicillin process given above Symbols !, &, | and <-> represent logical not, and, or and equivalence respectively. The symbol AG, meaning 'in all execution paths always', belongs to CTL [2] which is the logic used to specify properties in the SPEC section of the finite state machine. The SPEC section (lines 11-15) describes that the situation, in which all the valves and the pumps for tank1 are off, is equivalent to the situation, in which the valves VA, vA, VB, vB and the pump mA are off. That implies that the state of the pump mB can be eliminated in the operating interface if all the valves and the pumps for tank1 are off. In a state of the finite state machine, every boolean expression has a value. Examples of boolean expressions in the state of the finite state machine are:

!VA - 'VA is false' or 'Ellis lock VA is turned off'

Lev>0 - 'the liquid level of the tank is greater than 0'

ob - 'operator b is performing a remote action'

The transition relation of the finite state machine is defined by a relationship between the values of boolean expressions in a state and their values in the next state. The lines 34 to 37 describes the behaviour of the liquid level of the tanks. The lines 38 and 39 describe the behaviour of the inlet flow F1 of the two tanks. The lines 40 and 41 describe the behaviour of the outlet flow F2 of the two tanks. Consider the lines 43 and 44:

```
43    next(VA):=case LL:0;oa&!ob&!VA&
44      !mA&!vA&!VB&!mB&!vB:1;1:VA;esac;
```

This states that, if the level of the liquid is low in a state (LL=true), VA is turned off, i.e. next(VA)=0. This is an automatic safety interlock mechanism as in [6]). If operator a is performing a remote action (oa=true) whilst operator b is not (ob=false), and VA, mA, vA VB, mB and vB are turned off, then VA will be turned on in the next state (next(VA)=1). This is described in the code by (oa&!ob&!VA&!mA&!vAVB&!mB&!vB:1). Otherwise, the value of VA does not change (1:VA). Similarly, the lines 45 to 61 describe the behaviour of the valves VB, vA, vB, and the pumps mA and mB. The specification assumes that when an operator performs an action, as many components as possible are affected (e.g. if the conditions are right to open VA, mA and vA corresponding to both tank1 and tank2, then operator a will open all components). Other operator behavior could be specified as well.

The original description of the system refers to 2 tanks and components relating to a particular tank, e.g. tank1.VA or tank2.mB. In fact, the system subdivides naturally into two subsystems of components, those associated with tank1 and the others associated with tank2. This is specified in a modular fashion by defining a general `tank_subsystem` (lines 17-61) and then creating two instances tank1 and tank2 (lines 5 and 6) as the system. The corresponding pair of finite state machines executes synchronously, but the subdivision gives a more structured specification and, more importantly, is used to optimize the algorithm for generating candidate interfaces. This is discussed in 3.3 below.

3.3 Interface generation

To calculate the minimal interfaces for operator a, a brute force application of the algorithm in section 2.2 could require a call of NuSMV for every subset of every set of states of components in *Ua* which contain the components in Wa. This amounts to about 350 calls of NuSMV, with up to 12 components appearing in formulae to be verified or refuted by each call. However, this problem of size, occurring with large-scale systems, can be mitigated as they usually admit a natural decomposition into subsystems. Modular decomposition of systems has been used to avoid the state explosion problem associated with model checking [3]. There are further benefits when model checking is used for exhaustive generation of interfaces as proposed here, in that it reduces considerably the number of calls that are required of the model checker in the first place. Consider the set Ra. Modular specification of the system behaviour partitions components into 2 sets:

```
Ra1 = {tank1.VA,tank1.mA,tank1.vA,
       tank1.VB,tank1.mB,tank1.vB,
Ra2 = {tank2.VA,tank2.mA,tank2.vA,
       tank2.VB,tank2.mB,tank2.vB}
```

It is only the operator actions oa and ob that affect components in both sets, but their actions on each do not influence their actions on the other. Thus, the components in Ra1 and Ra2 are independent. Therefore, the minimal interfaces for Ra are precisely unions of minimal interfaces for Ra1 and Ra2. By symmetry, we need only consider Ra1. Now, minimal interfaces for Ra1 need to include the components to be changed, i.e.

```
{tank1.VA,tank1.mA,tank1.vA}
```

and minimal sets out of:

```
{tank1.VB},{tank1.mB},{tank1.vB},
{tank1.VB,tank1.mB},
{tank1.VB,tank1.vB},
```

```
{tank1.mB,tank1.vB},
(tank1.VB,tank1.mB,tank1.vB}
```

Hence, at most 7 calls of NuSMV are required. In fact, after executing NuSMV 6 times it is found that:

```
{tank1.VB,tank1.vB}
```

is, in this case, the unique minimal set of information in addition to {tank1.VA,tank1.mA,tank1.vA} required by operator a for the tank1 subsystem. It is illustrated in the SPEC section (lines 11 to 15). It is slightly surprising that information about the pump mB, which may suddenly stop irrespective of other components, is not required and can be inferred from the current state of other components. If a more straightforward safety interlock system as in [6] was adopted, we would find that any of the sets:

```
{tank1.VB}, {tank1.mB}, {tank1.vB}
```

would be minimal amounts of additional information. In this case, the list of minimal interfaces for operator a would be the list of sets which are the union of the set:

```
{tank1.VA,tank1.mA,tank1.vA,
 tank1.VA,tank2.mA,tank2.vA}
```

and each of the sets:

```
{tank1.VB,tank2.VB}
{tank1.VB,tank2.mB}
{tank1.VB,tank2.vB}
{tank1.mB,tank2.VB}
{tank1.mB,tank2.mB}
{tank1.mB,tank2.vB}
{tank1.vB,tank2.VB}
{tank1.vB,tank2.mB}
{tank1.vB,tank2.vB}
```

It is clear that decomposing the specification of the system has yielded a collection of candidate minimal interface options as essentially a cartesian product of sets of choices of components to be added at the interface.

3.4 Concurrency

If the pump mB breaks down, the initially specified interface requires operator a to be informed of this change in state of the offending pump. However, with the reduced interface, operator a does not need to be informed of the state of mB immediately, as information about VB and vB, arriving later, will suffice. This allows more scope for concurrency in the implementation of schedulers. A formal specification of such a scheduler can be produced by specifying a consistency condition on the concurrent reads and writes corresponding to the operator reading the state of components at the chosen minimal interface and changing the state of components respectively. The relevant consistency condition is that of serializability of uninterpreted transactions. The case of finitely many reads and writes is analyzed in [5]. A condition on an infinite continuous stream of reads and writes to be serializable is given in a temporal context in [4].

4 Conclusions

Formal methods have been used in several areas of human-computer interaction, including cognitive modelling and task analysis. The main goal of this work is to show that, for developing concurrent interfaces to large-scale systems, the problem of merely the *amount* of information to be presented at the interfaces is a major concern whose analysis presents a suitable role for formal methods. In this paper, it has been shown that the benefits of using a formal approach include clarification of the use of the interface, and analyses which suggest novel choices for interfaces that might not have been evident in an informal approach. The formal techniques for process plant control are based on the emerging use of temporal logic model checkers in such plants. The analogy with databases means that recent developments in database concurrency and model checking raise the possibility of an integrated formal development process, spanning the development of process plant control logic, concurrent operator interface design and scheduling. In general, remote operation over the Internet will be important in the future not only for process plant control, and the amount of information to be presented at interfaces will be a significant part of overall interface design.

Acknowledgement

This work was financially supported by the Royal Society in the UK through the Research Grants Scheme 2004/R1.

References

[1] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri, "NuSMV: a new symbolic model verifier", Proceedings of the International Conference on Computer-Aided Verification, *Lecture Notes in Computer Science*, No. 1633, Trento, Italy, pp. 495-499, Jul. 1999.

[2] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications", ACM *Transactions on Programming Languages and Systems*, Vol. 8, No. 2, pp. 244-263, 1986.

[3] O. Grumbler, and D. E. Long, "Model checking and modular verification", *ACM Transactions on*

Programming Languages and Systems, Vol. 16, No. 3, pp. 843-871, May 1994.

[4] W. Hussak, "Serializable histories in quantified propositional temporal logic", *International Journal of Computer Mathematics*, (to appear 2004).

[5] C. H. Papadimitriou, "The serializability of concurrent database updates", *Journal of the Association for Computing Machinery*, Vol. 26, No. 4, pp. 631-653, Oct. 1979.

[6] S. H. Yang, L. S. Tan, and C. H. He, "Automatic verification of safety interlock systems for industrial processes", *Journal of loss prevention in the process industries*, Vol. 14, pp. 379-386, 2001.

[7] S.H. Yang, X. Zuo, and L. Yang, "Control system design for Internet-enabled arm robots", Lecture Notes in AI, Vol. 3029, pp. 663-672, 2004.