F. Wagner February 2006

# Completeness of information in the virtual environment - the “unknown” (“undefined”) control values

## *The problem*

Let's begin by re-stating the basic concept of VFSM relating to the topic:

- The entire control function is based on a virtual input which reflects the full (control) information about the relevant inputs.
- If that rule is fulfilled we are able to specify one or more state machines realising a control function and
- we can do it without the necessity to “complete” later the specification by some fixes in the code.

By definition, the fixes in the code are impossible using StateWORKS tools and run-time system as there is no code generated.

Now let's concentrate on the term “full control information”. That term means that we operate with input names which describe all imaginable situations which may arise on inputs. Let's then take the simplest input signal used in control: the digital input. It may have two defined values which are named: true/false or on/off or HIGH/LOW, to cite the most often used. The classical boolean algebra is based on those values. What do we do using boolean algebra if we do not know the value? We solve the situation by:

- setting the boolean variable to a default value: true, false or we leave the last one depending on the situation and
- handling that situation as a special case.

## *“Unknown” control values*

Using StateWORKS we cannot code that situation as a special case: it must be a normal case treated exactly in the same way as the situations: HIGH or LOW. The solution is to introduce a third “unknown” control value which gets the name: UNKNOWN in addition to the two values: HIGH and LOW.

We shall show you the consequences of that concept by modifying the state machine Pressure of which a simplified version has been described in the case study StateWORKSForIndustrialControl.pdf[1]. The project Pressure contains the details of the state machine and you may download it from our site. For your convenience Figure 1 shows the Pressure state transition diagram.

The state machine Pressure uses one digital input called just Di. We used only two possible control values: HIGH and LOW creating the names: Pump_Ok and Pump_TooHot on them. Why didn't we use the third possible control value: UNKNOWN? Well, we just ignored that case but maybe we were too careless.

Do we really have a need to discuss such a “unknown” situation. The signal comes from a sensor which produces it if the pump temperature is getting too high. There is a relatively long and unreliable path from the sensor to a control system: the sensor itself, electronics, a cable,

1 The requirements are expanded by a demand for cooperation with a Master state machine. Therefore the state machine Pressure used differs slightly from the state machine in the referenced technical note. Because the differences are irrelevant to the discussed topic we do not go into details.

connectors. So, the probability that something goes wrong can not be neglected. What will happen in case of a malfunction in the system? Well, this situation is ignored, the system does not know about it. A good design should provide means to supply information about any erroneous situation. If it is the case we can use the control value UNKNOWN. Let's do it! So, we define a name Pump_Unknown. ( A better name might be Pump_Status_Unknown, in fact.) That's easy but how can we use it in the state machine specification?

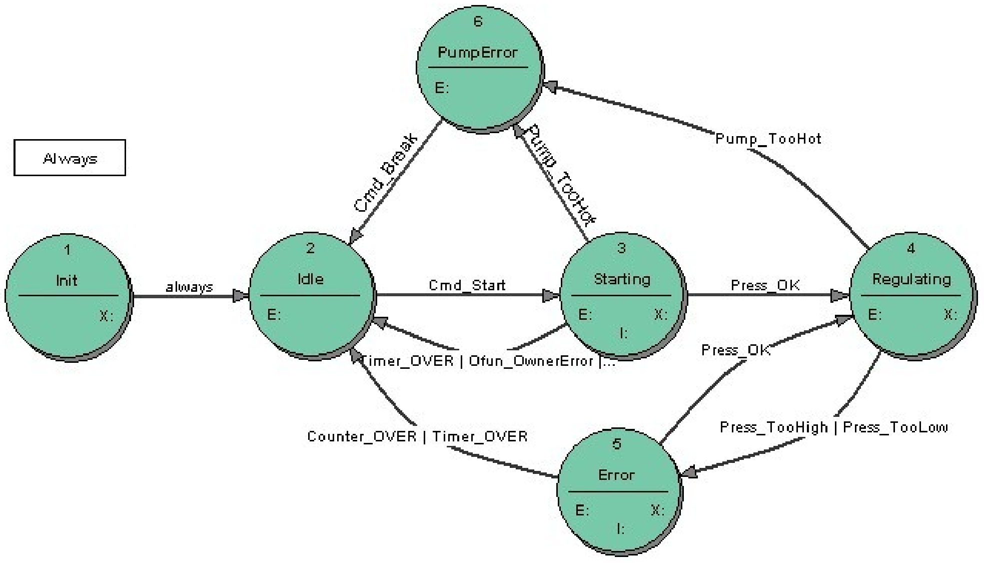

*Figure 1 Pressure: the state transition diagram*

First of all we have to extend the requirements specifying what is expected from the control system if the signal from the Pump temperature sensor is not delivered any more: We decide that *an alarm should be issued and the system should be given a chance to repair that situation* (maybe it will work if we touch or shake the connector).

We need:

- An additional Input name on the UNKNOWN control value of the digital input Di. Let's call it Pump_Unknown.
- An alarm. Let's call it Al_PumpUnknown specifying Output names on Coming and Going.
- A timer to delay the transition to the state Idle. For sake of simplicity we use the existing timer instead of introducing a new one.

How to implement the control? If we think about it we will see several solutions; most of them are erroneous. We could analyse some, at the first glance, “obvious” but after more thorough analysis erroneous, solutions. Eventually, one would probably decide to add two additional states: Pump_UnknownStart and Pump_UnknownReg. That solution has been implemented in the project Pressure1. Figure 3 shows the Pressure1 state transition diagram.

We have showed you a direct use of the “unknown” control value. Also an indirect use of that value make sense sometimes. This is useful and required in situations when you have in the same state transition or action conditions using both digital control values: HIGH and LOW. How to ignore them, so that the conditions where those values are used cannot be fulfilled? The answer is: the existence of the “unknown” control values solves automatically the problem.

## *“Undefined” control values*

Which inputs (in other words, objects in the RTDB) should have the “unknown” control values?

The answer is: all inputs which are not reliable, i.e. inputs that are generated outside the computer. It would not make much sense to define for instance an "unknown" value for a timer. If a timer does not work it is a programming error and not a control information: we do not control software bugs!

To expand the idea of "unknown" ("undefined") control values we would like to discuss now the "unknown" control value of a parameter. In that case we gave it the name UNDEF believing that it expresses a bit better its meaning: the parameter value has been not properly defined. It may happen if the parameter should be loaded from a file but the file does not exist (because we deleted it, for instance, by mistake).

The PAR object is used to store data but it exhibits also some behaviour as described by the state transition diagram in Figure 2.

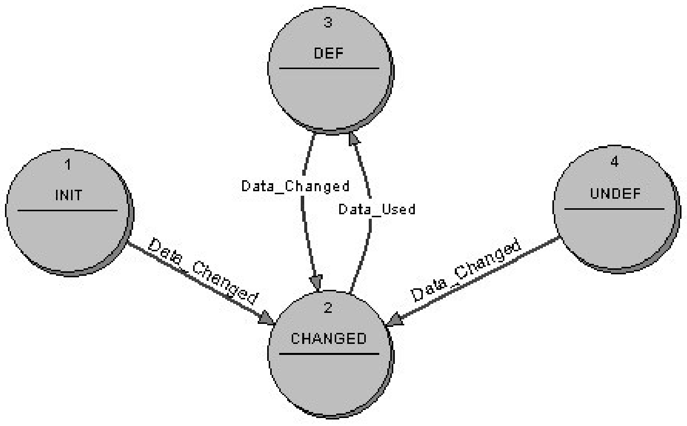

*Figure 2 The state transition diagram for the PAR object*

Let's assume that while starting the system the object PAR is created and initialised to a state UNDEF. This state signals that the PAR data equals the default value specified in the project.

*Now imagine that we wanted the required pressure value to be taken from a file, otherwise the system should wait for a while. When the timer elapses the system should generate an alarm that the Pressure value is not correct and continue to wait until the parameter data changes.*

In the project Pressure the required pressure value is defined by the object Par:RequiredPressure. We use already in the state machine Pressure the changes of that object to trigger the call of the output function calculating the Pressure limits for the Swip object. If in the start-up phase the data is loaded from a file to the Par:RequiredPressure (for instance in the I/O unit) the state of the object will change to CHANGED and if it is used immediately to DEF.

The possible implementation requires:

- An additional Input name on UNDEF control value of the parameter Par (object Par:RequiredPressure). Alternatively, instead of using the UNDEF control value we may do it indirectly using the DEF control value which we have called RequiredPressure_DEF (the reason for this decision can be seen by studying the state ParUndef in the project).
- An alarm. Let's call it Al_ReqPressureUndef_Coming specifying Output names on Coming and Going.
- A timer to delay the transition to the state Idle. Let's call it TiInit..
- an additional state ParUndef.

Figure 3 shows the state transition diagram of the state machine Pressure1 with changes reflecting both discussed extensions: the use of the control value UNKNOWN for the digital input and the use of the control value UNDEF for the parameter object. In the specification of the state machine Pressure1 we use indirectly the control value UNDEF: the state machine is initialised to that state but we never used it in the transition conditions.

The described use of UNDEF system requires that the PAR object will be of a PP type. If you choose the EP type the object will be initialised to the state Init and will never reach the state DEF. The sense of that arrangement exceeds the scope of this technical note; we leave it for another occasion.

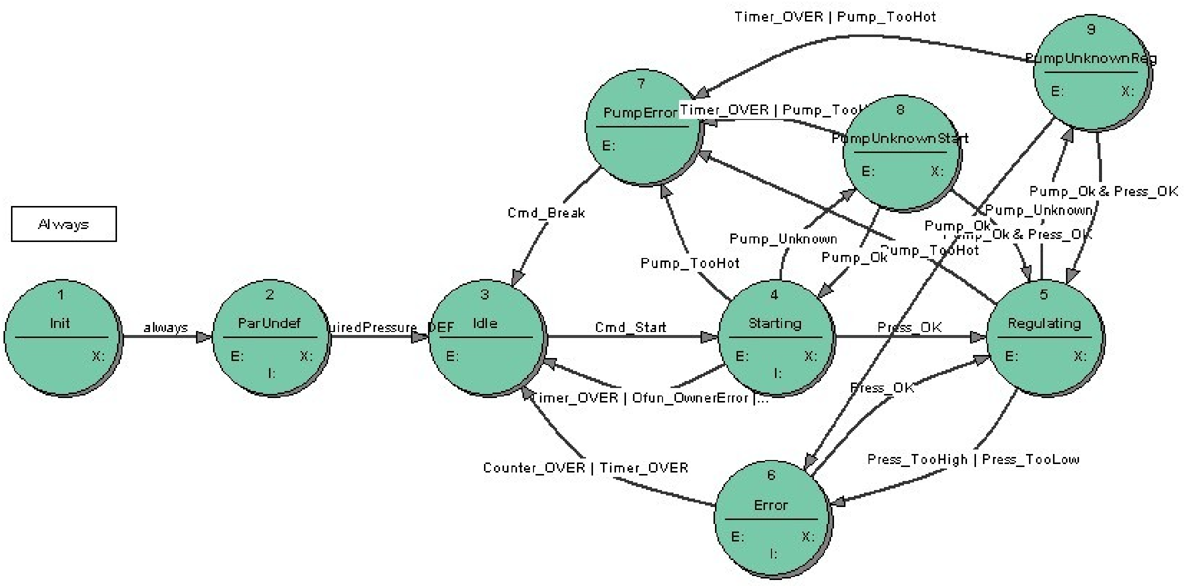

*Figure 3 Pressure1: the state transition diagram*

Objects: DAT and NI exhibit a similar behaviour to a PAR object in being initialised to a state UNDEF. In addition, that control value is also passed to any SWIP object which supervises the data value of an object with the UNDEF control value. If such an object is in the UNDEF state the supervising SWIP object sets also its UNDEF control value.

## Conclusions

The idea of "unknown" control value is a must for modelling systems that are to cover the entire system behaviour. Eventually, if completed by objects like XDA or even further by OFUN it allows any input situation be described by suitable control values.

StateWORKS is a system, which realises the idea of an executable specification. Since it does not use a code generation step, the specification must be complete and any changes and removals of errors (which can only be logical, rather than coding errors) can be done only in the specification.