Tcl/Tk based Equipment Station Control in a Semiconductor Wafer Fabrication Area

Author: Campbell Boyd

Company: National Semiconductor (UK) Ltd.

Abstract:

In the mid-1980s, most semiconductor manufacturers purchased Manufacturing Execution Systems, such as WorkStream or Promis, and installed processing equipment in their wafer fabs that had SECS-II or other interfaces. The MES contained the configuration data that defined correct processing but without some kind of link to the equipment via the interface this could not be enforced. Ensuring correct processing by, for example, automating recipe selection, lowers costs by improving yields. Automating the links to the processing equipment and to the MES provides opportunities to improve process control and productivity, thus further reducing the cost of manufacturing. The chosen concept to provide these links is that of a Station Controller (also known as a Cell Controller). The Station Controller provides the operator with a GUI that contains all information and control functions that are required to correctly process wafers through a piece of equipment.

This paper describes a set of easily modifiable and adaptable Tcl/Tk applications that are used as Station Controllers in the wafer fabrication area. The applications run on desktop or industrial PCs that are linked to equipment such as epitaxial reactors, ion implanters or furnaces. Each Station Controller can be used for one or more machines that are used at a bay or cell. The main GUI implements an object-oriented methodology using [incr Tcl] to encapsulate each machine from any other used on the same Station Controller.

To control the overall process, a State Diagram is employed where the logical flow can be altered by interrogating the machine via an interface, input from the operator, delays or by other methods defined in the main GUI objects. The State Diagram can be edited by a Tk canvas application that makes the diagram look like a flow chart. Thus, the flow diagram for the process and the code that implements it, are, in effect, one and the same. Similar state diagrams are also used for interfaces to the equipment and these can be modified and reloaded without interrupting the main application thus speeding development time. Typically, the interface state diagrams build Tcl lists that contain valid SECS-II commands or interpret returned messages. The interface uses DMH software to bridge between Tcl & SECS-II. An editable real-time database stores current values of critical variables in case of interruptions on the PC. The link to the plant MES, WorkStream, uses Expect to automate the logging of transactions via 2 Linux servers that each run several emulated terminal sessions. The Station Controllers also have a interface to a Real Time Dispatching system from Brooks PRI and links to the plant intranet.

The Station Controllers are a cost-effective solution to improving yields and throughput and of reducing costs in a wafer fabrication area. Since they are based almost entirely on open source software, there are almost no licensing costs associated with implementing them. Since they have been developed in-house, they can be continually modified and improved to meet ever-changing business conditions. They offer much greater flexibility and better integration with other plant systems than could ever be achieved by buying off-the-shelf solutions which would not provide the same look & feel across different kinds of equipment that the Station Controllers do.

Introduction

Wafer fabrication is the initial manufacturing stage of the semiconductor industry. It produces silicon wafers each of which may contain hundreds to possibly several thousands of the fully functional cores of a given design of integrated circuit, or product. At later stages, the wafers are sliced and diced to separate the circuits (or chips as they are colloquially called) and known good ones are assembled and tested ready for sale.

In a wafer fabrication area (or wafer fab, for short), 25 or so wafers will be processed together as a lot from start to finish. Each wafer in the lot will be of the same product and the whole process may contain several hundred processing steps or operations. These steps use a wide variety of processing equipment that utilise a extensive range of physical and chemical conditions. In a typical wafer fab, there will only be a dozen or so pieces of equipment of a given type such as ion implanters, sputter reactors, steppers, aligners etc. Because the integrated circuits are manufactured by successively building layers, each of which is similar in many respects, this implies that any given lot will pass through the same kind of equipment several times while in fab. The complexity of flows, products and varying equipment conditions means that semiconductor producers have had to use computer-based tracking systems such as WorkStream or Promis to track lots, record data and control production. These systems are often referred to as Manufacturing Execution Systems (MES). See Figure 1: Simplified Overview of Semiconductor Wafer Fabrication.

The MES will contain the configuration data for all given products that can be manufactured in that area. At any operation, for any product, the correct steps and recipes to be used will be specified in some manner. Typically, however, the actual recipes will be on the processing equipment since the details of these vary greatly from equipment type to type. Since each equipment type typically performs physically different processes from other types, the details of its operation are usually completely different. To illustrate this point with a simple example, some equipment like photoresist tracks process a single wafer at a time. Others, such as implanters or epitaxial reactors, process part (say 6 to 8 wafers) of a lot at a time while others like furnaces can process between one or six whole lots at a time. A consequence of these differences is that details of the human interface to the equipment are often bewilderingly different. This makes it time consuming to train operators to use a given piece of equipment and difficult for them to transfer from type to type. An interface with a common 'look & feel' would ease this problem.

To ensure correct processing at a particular step, it is critical that the correct recipe, if applicable, is used. To provide manufacturing flexibility, each piece of equipment may be qualified for several different processes. If human operators key in recipe names to the equipment based on information presented to them from the MES and they make a mistake, the consequences can be financially horrific. Each wafer might represent a potential of $10,000 or more of revenue to the company and there are processes where more than 100 wafers are processed at a time, for example, at a furnace. A single character typing error (eg. entering ID06 instead of ID07) could result in a potential loss exceeding $1,000,000. The loss is not only financial since another lot or lots may need to be started to replace the scrapped ones and this could mean not fulfilling an order in time to a critical customer who can often purchase equivalent parts from other competitor companies.

One way of avoiding these serious problems, is to link the MES and the processing equipment so that, among other things, it is possible to download the correct recipe name direct to the equipment. To facilitate this, the Semiconductor Equipment and Materials Institute (SEMI) produced the SECS-II standard for computer interfaces to fab processing equipment. However, like the equipment itself, the details of these interfaces are often fairly different from equipment type to type and the sequence of instructions needed to perform a recipe id download, for example, are rarely identical. The manuals that accompany these interfaces are not always totally complete and accurate. Constructing an interface is often a matter or trial and error so that a system where code can be changed quickly and instantly retested would be of great value. Tcl is ideal for this since no compilation is necessary and pieces of code can be sourced when needed.

The Station Controllers meet the needs just outlined. They are quick to develop, test and modify. By using libraries of common code and Object Oriented methods brand new prototypes can be made functional in a few days. This also ensures a common 'look and feel'. Most importantly they help reduce costs while being cheap to implement.

External Interfaces

As outlined in the introduction, one of the advantages of the Station Controller concept is the capability to bring information from different external sources to the one GUI on the Controller. See Figure 2: Station Controller Concept. Two of the most important, in this application, are the Dispatch system and the MES, WorkStream. These two are now described.

Dispatch System

At NSUK, the Real Time Dispatch (RTD) system from Brooks PRI's APF family is used. This runs on 2 independent Sun servers in separate computer rooms to provide robustness and resiliency. See Figure 9: MES Interface Architecture. A dispatch list is a list of lots in descending order of priority. The purpose of the lists is to improve cycle time by presenting the recommended lots the operator should choose to process. In addition, high priority lots for product or process qualifications can be clearly identified and equipment utilisation can be optimised by grouping lots with identical recipes together. In the RTD system, production personnel control the rules which determine the prioritisation and batching of lots by using the graphical RTD rule editor. The Station Controller receives the dispatch list by remotely executing commands on one of the RTD Sun servers. If the first is not available, the second will be tried automatically.

The output from one of these remote commands looks like this:-

```
#headers
'STATION: NOVA-1     MOVEABLE LOTS:  26 QTY:  573     HELD LOTS:   0 QTY:    0     BLOCKED LOTS:   0 QTY:    0'
#columns  'Lot Number' 'Name' 'Hold' 'Hot' 'Own' 'Qty' ' Oper' ' Description' 'Tech' 'Code' ' CTPML' 'Static' 'Moved In' 'Runing
On' 'Species'
#widths  12 10 4 4 3 3 5 16 8 8 6 6 8 9 12
#types 'STRING' 'STRING' 'STRING' 'STRING' 'STRING' 'INTEGER' 'INTEGER' 'STRING' 'STRING' 'STRING' 'REAL' 'STRING' 'STRING'
'STRING' 'STRING'
#lotcol 0
#rows
'JM02C456' 'WERNER' ' ' 'HOT' 'P' '22' '5550' ' EMT IP PNP' 'VIP1+' 'IB-326' '2.15' ' 0.2' '  NO' '' ''
'JM02C521' 'FINLAND' ' ' 'HOT' 'P' '24' '5550' ' EMT IP PNP' 'VIP1+' 'IB-326' '2.00' ' 0.2' '  NO' '' ''
'BOBTEST10' 'BOBTEST10' ' ' '' 'E' '0' '5440' ' BASE IMP' 'SLM' 'IB-341' '25.67' ' 208.0' '  YES' 'DIMPLE' 'BORON'
'JOHNTEST99' 'JOHNTEST99' ' ' '' 'E' '0' '5440' ' BASE IMP' 'SLM' 'IB-341' '4.80' ' 21.9' '  YES' '' ''
'JM02C772' 'NATALIE' ' ' '' 'P' '24' '5440' ' BASE IMP' 'SLM' 'IB-345' '2.50' ' 0.2' '  NO' '' ''
'JM02C647' 'INTERPOL' ' ' '' 'P' '23' '3755' ' P+BL IMP' 'AMPS' 'IB-325' '2.38' ' 0.2' '  YES' '' ''
'JM02C650' 'BARBOUR' ' ' '' 'P' '24' '5440' ' BASE IMP' 'SLM' 'IB-346' '2.35' ' 0.1' '  NO' '' ''
'JM02C781' 'ACAPULCO' ' ' '' 'P' '24' '5440' ' BASE IMP' 'SLM' 'IB-341' '2.33' ' 0.1' '  NO' '' ''
'JM02C776' 'SUMMERS' ' ' '' 'P' '24' '5440' ' BASE IMP' 'SLM' 'IB-291' '2.33' ' 0.1' '  YES' 'NOVA-6' 'BORON'
'JM02C838' 'BURTON' ' ' '' 'P' '24' '5076' ' UP ISO IMP' 'VIP1+' 'IB-290' '2.32' ' 0.0' '  NO' '' ''
'JM02C414' 'LILLIAN' ' ' '' 'P' '24' '5550' ' EMT IP PNP' 'VIP1+' 'IB-326' '2.22' ' 1.9' '  YES' 'NOVA-1' 'BORON'
'JM02C835' 'TIMON' ' ' '' 'P' '24' '5440' ' BASE IMP' 'SLM' 'IB-345' '2.21' ' 0.2' '  NO' '' ''
'JM02C890' 'FORBES' ' ' '' 'P' '24' '5076' ' UP ISO IMP' 'VIP1+' 'IB-290' '2.18' ' 0.1' '  NO' '' ''
'JM02C766' 'ONICH' ' ' '' 'P' '24' '5205' ' ISO/DB IMP' 'LB250' 'IB-328' '2.17' ' 0.4' '  NO' '' ''
'JM02C451' 'PTOLOMEY' ' ' '' 'P' '24' '5550' ' EMT IP PNP' 'VIP1+' 'IB-326' '2.15' ' 1.9' '  NO' '' ''
'JM02C452' 'CARLIN' ' ' '' 'P' '24' '5550' ' EMT IP PNP' 'VIP1+' 'IB-326' '2.15' ' 0.2' '  NO' '' ''
'JM02C453' 'FIELDS' ' ' '' 'P' '24' '5550' ' EMT IP PNP' 'VIP1+' 'IB-326' '2.15' ' 0.2' '  NO' '' ''
'JM02C454' 'RASTUS' ' ' '' 'P' '24' '5550' ' EMT IP PNP' 'VIP1+' 'IB-326' '2.15' ' 1.9' '  NO' '' ''
'JM02C455' 'SYKES' ' ' '' 'P' '24' '5550' ' EMT IP PNP' 'VIP1+' 'IB-326' '2.15' ' 1.9' '  NO' '' ''
'JM02C782' 'PAUL' ' ' '' 'P' '24' '5440' ' BASE IMP' 'SLM' 'IB-342' '2.10' ' 0.2' '  NO' '' ''
'JM02C635' 'SKIRVING' ' ' '' 'P' '24' '3280' ' HE IMPL' 'LMDMOS' 'IB-294' '2.10' ' 0.1' '  YES' '' ''
'JM02C639' 'BASSETT' ' ' '' 'P' '24' '5440' ' BASE IMP' 'SLM' 'IB-345' '2.07' ' 0.1' '  NO' '' ''
'JM02C745' 'DYLAN' ' ' '' 'P' '24' '5440' ' BASE IMP' 'SLM' 'IB-344' '1.98' ' 0.2' '  NO' '' ''
'JM02C725' 'CHICAGO' ' ' '' 'P' '24' '5440' ' BASE IMP' 'SLM' 'IB-341' '1.94' ' 0.1' '  NO' '' ''
'JY02C751' 'KILLIN' ' ' '' 'P' '24' '5440' ' BASE IMP' 'BIFET' 'IB-342' '1.83' ' 0.4' '  NO' '' ''
'JM02C742' 'COULTER' ' ' '' 'P' '24' '5440' ' BASE IMP' 'SLM' 'IB-291' '1.78' ' 0.2' '  NO' '' ''

#blocked '' '' '' '' '' '' '' '' '' '' '' '' '' '' '' '' '' '' '' '' '' '' '' '' '' '' '' '' '' '' ''
#batch  7 7 1 1 2 9 3 1 4 11 7 2 11 10 7 7 7 7 7 6 8 2 5 1 6 4
#ISS-DLIS-DIAGS|FAB3|SchedDB3|NOVA-1|IMPLANT|STATION|0|16233264|22/05/02 14:25:26|0.000100017|1.83749|0|26|0|0|N|||||||||||||
```

WorkStream Interface

The MES in use at NSUK is WorkStream from Applied Material's Consilium subsidiary. This is a 20-year old application that historically has been accessed by VT terminals via text-based forms that have 4-character mnemonic

codes. (A couple of examples of WorkStream screens are shown in the Illustrative Examples). At some processes, the operator has had to spend a significant portion of time, entering (by hand typing) or receiving information from WorkStream. One advantage of the Station Controller is the ability to automate this to some extent and to replace the text-based interface. When some action that requires sending or receiving data from WorkStream is initiated from the Station Controller, all the data that would formerly have had to be hand entered (eg. Lot numbers, equipment identification, event names) can be sent direct to the WorkStream interface without the operator having to type every character.

Two Linux servers (again in separate computer rooms as with the RTD servers) run several Expect sessions which await requests from any Station Controller in the fab area. The requests are transmitted using the http protocol and initiate cgi scripts on the chosen Linux server. The cgi script calls the relevant Tcl script for the particular request corresponding to one of WorkStream's form codes. These scripts use Expect to emulate a human hand-entering the received data from the Station Controller into WorkStream. If the request was to update WorkStream, either a success message or the error message that would be displayed on a VT terminal is relayed back to the Station Controller. If the request was to obtain data from WorkStream, then the useful data is extracted from the WorkStream screen, packaged as a Tcl list and returned to the Station Controller where it may be further parsed for display or to determine a course of action. See Figure 9:MES Interface Architecture.

If additional WorkStream functions are needed, this is straightforward. The layout and form name of a given WorkStream screen are examined and entered into a renamed copy of an existing interface Tcl script. Similarly a new cgi script is created and changed. The httphostif library script, common to all Station Controllers, is edited to add a new method that will call the new interface scripts. The new function can be tested independently of a running Station Controller using a debug version of the interface which is in effect a Tcl-based VT terminal emulator. The list that would be received from a Station Controller is entered as a command line, the emulator displays what WorkStream would do and the list that would be returned to the Controller is displayed. The function can be refined until it performs as desired including handling any WorkStream error messages that might occur.

Since the interface uses the http protocol, it will be no surprise that the interface and its performance are accessed via web pages. It can be stopped, started and reloaded by clicking links in a browser. All messages being received and returned are viewed on a browser. If there is some problem with WorkStream itself, this can be quickly identified. For example, if one of the VAX computers that run WorkStream goes down, the interface server automatically attempts to log any affected Expect sessions onto another, working, VAX.

Typical Components of a Station Controller

Typically, a Station Controller runs as 3 processes, namely the Real Time Database, the SECSII driver and a main application, all connected by the comm package. All 3 are started by a single script which starts them in the sequence just given. The following descriptions also follow this order.

Real Time Database

This has two main purposes. The first is to preserve the values of important variables so that in the event of power disruptions, inadvertent or deliberate reboots etc., the overall state of the Station Controller can be restored to the same condition prior to the interruption. The second is to serve as a repository of any desired variables for retrieval, display or manipulation elsewhere in the overall system without explicitly having to have methods of passing variables from object to object. The variables to be stored in the RTDB are defined in a text-editable configuration file that is read at start up. Another feature is a tool to examine (and edit if necessary) the actual values in the database which is useful when debugging a new Station Controller.

Equipment Interface

When a SECSII interface is available on a piece of equipment, a driver process is used to send and receive messages from the tool. The actual driver code is common to all Station Controllers but a configuration file is used to specify

which interface components are run continuously and which are called on demand from the main application. Usually, those that are run continuously are ones which perform a handshake with the equipment every 30 secs or so or which request and receive equipment status or alarms. A component run on demand might be one like a recipe download function. Where a SECSII interface is not available, a DDE link is made to custom-built software that controls the equipment but otherwise the interface to the equipment is very similar.

All these components build SECSII messages which are almost always in a list format. (A page from a SECSII manual is given in one of the Illustrative Examples). Tcl is ideal for this. The actual interface is provided by portions of a package called DMH from Hume Integration which was specifically written to provide a Tcl to SECSII interface.

During development, the code for a component can be altered and reloaded without affecting or restarting any other part of the Station Controller. The effect of the change can be seen and the response from the equipment examined until the developer is satisfied the component meets requirements.

Main Application

This is started by reading a configuration file and sourcing any desired packages and libraries that will be utilised in the application. As it starts, links to the RTDB, driver and MES interface are established. A frame is then created which is common to all Station Controllers. This contains portions displaying date and time, a user id, a status bar and buttons to log out or exit the application. Within this frame a handful of objects are created which form the main functions the equipment operator or others will use. These objects utilise a BLT tab set to separate them visually. If the Station Controller is used for more than one piece of equipment, there will be a separate set of functional tabs for each piece of equipment. Colours, defined in the configuration file, are used to codify the set of tabs that belong to a given tool.

The kinds of functional tabs that most Station Controllers have include a main Production one which is usually displayed by default on top of the others since it will be the most used. More details on how this is used are given below. See Figure 3: A typical Station Controller Screen. A Status tab allows the operator to view full details of the equipment's status from WorkStream and also allows the logging of other events, defined in the application configuration file. This Status object's methods are also called elsewhere in the main application whenever the equipment's status is required. This is illustrated in one of the examples below. The only other common tabs are one which utilises Internet Explorer to allow restricted access to the plant's intranet and another which displays scheduled activities. The IE web tab is used to supply help, troubleshooting and other sources of documentation of value at the process concerned. Access can also be given to other web-enabled applications such as Flextime, internal telephone book etc. Since these web pages are on a central server, none of the content is copied to the Station Controller and, of course, they can be updated without affecting the use of the Controller. The Scheduled activities tab displays scheduled events from WorkStream and work orders from IFS, NSUK's plant ERP.

Other kinds of tabs tend to be specific to the particular equipment or process. These include pages for special tasks carried out by engineers or technicians, electronic log sheets, run advice or functions unique to one process such as mask plate database queries.

State Diagrams

By and large, the main application scripts which create the functional tab sets, consist of nothing but code to inherit a standard class, modify this in a constructor and several methods that may be called as desired. What then provides the 'Control' implicit in the name Station Controller? This is accomplished by a State Diagram (see Figure 4: An Example of a State Diagram) which is constructed to control the entire process cycle from start to finish. At any point in time, the RTDB variable STATE is set to one and only one of the states in the diagram. The variable's value is changed by calling methods which can branch the flow after being evaluated. Time delayed loops can also be used. The State Diagram (as a file, see Figure 6:Extract from State Diagram file) is simply a set of Tcl lists but a special file extension (.sd) is used so that when being opened for edit, a special Tcl application, the State Diagram Editor is called. When so opened (see Figure 5:Updating Code in a State Diagram), a Tk canvas is created

configured by the lists in the file and where the various states appear as bubbles and they are connected to other state bubbles by flow arrows. New bubbles or arrows can be added or old ones deleted. Any of these can be selected and moved. The crucial point is that Tcl code can be inserted into the bubbles to call the methods (in the main application code) that will control the flow.

For example, a Start button on the main Production tab may move the state diagram from its initial state to a second one where there will be a call to a method which evaluates a set of checks to determine if the process can continue. These checks might include requesting the equipment status in WorkStream. The check would pass if the status was UP but in STANDBY, for example. The method will return, say, a 1 if all the checks pass and 0 otherwise. The Tcl code in the bubble determines if the returned value equates to 1 and moves to the next bubble in the flow if it does. The next bubble may initiate a recipe download. If it does not equate to 1, the flow will branch to a 'Start Checks Failed' bubble.

Another use of the same State Diagram editor creates flow charts for the SECSII interface. However where the main state diagram that controls the overall flow typically interacts with the operator or the WorkStream interface, the driver state diagrams interact with the equipment interface and pass messages back to the main application.

A further use of State Diagrams is for use with a Rules Engine. This is utilised at processes that have complex rules covering what kinds of test runs must be performed after planned equipment maintenance before the equipment is production ready.

Typical Use

The Production tab contains a dispatch list from the RTD system. See Figure 3: A typical Station Controller Screen. This can be refreshed as desired or automatically after a run is completed. It is also forced to be out of date if not refreshed after, say, 10 or 15 minutes. This list allows an operator to select one or more lots from it. The dispatch list will usually show lots that are ready to be loaded into the process tool. On selecting a lot, more details of the lot may be acquired from WorkStream, such as events to be logged at start and finish and details of engineering data to be collected. Other information may be requested from the operator, such as the identification of the mask plate to be used at a photo operation. If more than one lot can be loaded at one time, the operator can now select these too. However, to save time not all the checks or additional data will be performed again. Instead a quick check that the second and subsequent lots are at the same processing point as the first is sufficient.

The operator would now physically load the machine with the lot(s) selected. When this is completed, the Start button on the Station Controller is pressed. (A Station Controller screen displaying the Start button is shown in the Equipment Interface Illustrative Example). This initiates the various commands for a recipe download. More details of this are given in the first Illustrative Example. This usually entails checking in detail that the equipment is ready to be used, that no cleans or checks are overdue and of logging an event to WorkStream to indicate that the equipment is now in use. The recipe id obtained from the dispatch list or the lot inquiry will be sent via the SECSII interface to the equipment. The equipment typically returns an acknowledgement that the recipe is available and has been loaded. Usually, the operator has to start the equipment independently of the Station Controller. The Station Controller usually logs a start event on WorkStream at this point. The State Diagram will be in the Running State.

On some Station Controllers, the SECSII or other interface periodically updates the Controller with the equipment status automatically. For example, the status of the equipment will change from PROCESSING to COMPLETE when the recipe ends. On detecting this change, the Station Controller may be programmed to log an end event to WorkStream and automatically replace the recipe loaded when starting. Where the equipment interface cannot reliably return its status, the operator indicates the process has completed by pressing an End button on the Production tab.

Test wafers are often run with the production lots and these are removed after the run to have measurements made on them. In a furnace process, for example, the thickness of oxide grown during the run will be measured. The operator will press another button on the Station Controller and the results of these measurements will be entered. This data is then sent to WorkStream. At some processes, the metrology equipment has its own Station Controller

and once measurements have been made, it is the metrology Controller which sends the data back to the requesting process equipment Station Controller.

Once measurements have been entered, the process is complete. The Station Controller is reset (that is many of the variables such as lot numbers are set to null values), the dispatch list is refreshed and the State Diagram returns to its initial state. The whole process is now ready for the next lots to be selected.

Illustrative Examples

Recipe Download at Implant

In the main application, the lot selection from the dispatch list assigns values to RTDB variables like RECIPE & LOTNO. Then, when the recipe download is initiated, the following method is called.

```
method macroSetup {} {
      if [catch {deviceIF Recipe_Setup $RECIPE $LOTNO $WAFERS} msg] {
            Log_write ERROR "Recipe Setup Failed $msg"
            return -1
      }
      return 0
}
```

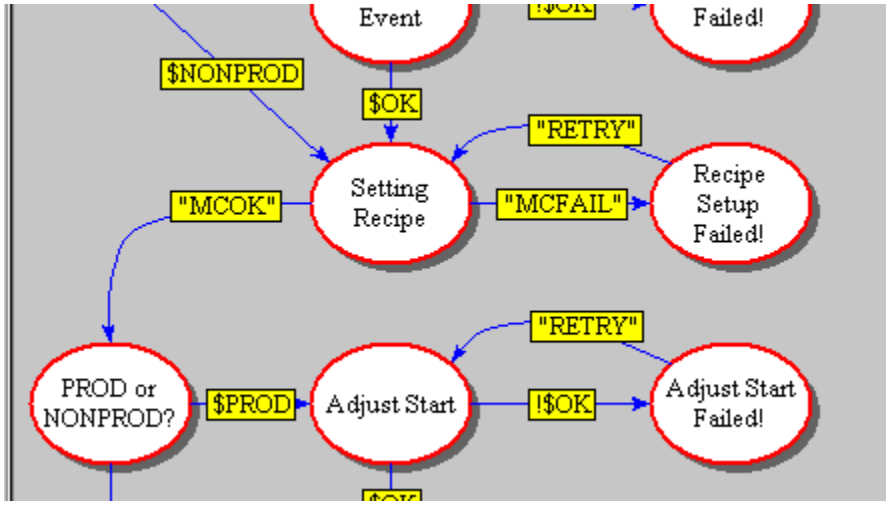
This passes these important values to the driver process. There the following piece of code starts up a driver State Diagram. Note that the interface has added a 'unit' variable. This is, in effect, an object reference so that the driver can pass messages back to the correct object that initiated the recipe download. The equipment interface (indeed all State Diagrams) are outside the Object Oriented environment of the main application.

```
proc Recipe_Setup {unit recipe lot count} {
      global RECIPE LOT COUNT
      set RECIPE $recipe
      set LOT $lot
      set COUNT $count
      puts "Selecting Recipe $RECIPE , $LOT , $COUNT"
      State_Machine_Start RecipeSetup
}
```

The RecipeSetup State Diagram and the crucial part of it that passes the recipe to the implanter's control system looks like Figure 8:Implant RecipeSetup State Diagram. The last bubble in this diagram sends a message indicating a successful recipe set up (or not) back to the main application. There, there is a method (see below) which passes a message to the main application's State Diagram (which is what provides the control for the Station Controller) to move it to the next state.

```
method SetupAcknowledge {msg} {
      if {$msg != "OK"} {
            Log_write ERROR "$entity Setup Failed !! - $msg"
            productionMgr sequencer "MCFAIL"
      } {
            Log_write INFO "$entity Setup Successful"
            productionMgr sequencer "MCOK"
      }
```

}



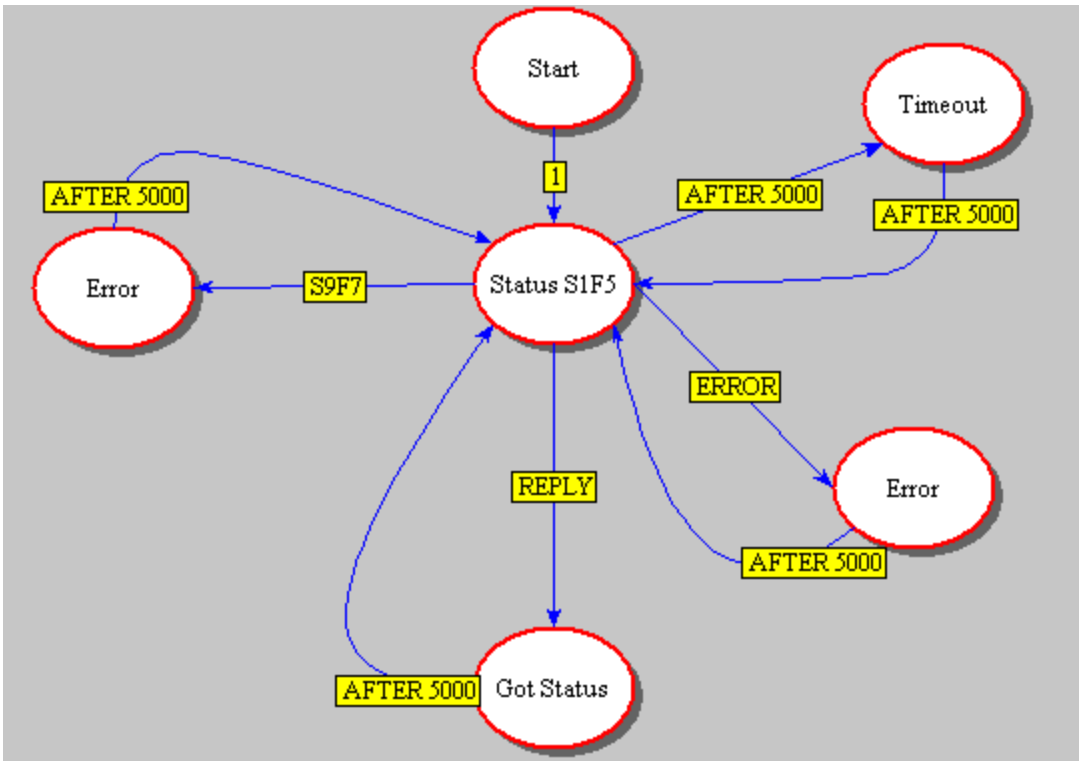
Equipment Interface on a Furnace Stack

A furnace stack consists of 4 tubes that can be heated up to 1300 °C arranged vertically above one another. (The photograph in the Equipment Interface box in Figure 2 shows a furnace stack). A single MUX computer provides a SECSII interface to the whole stack. Each tube is identified by a 'device-id' which is configured so that the top tube, which is conventionally tube 1, has device id 1, the tube below has id 2 etc. It is desirable to know the status of each tube at all times. The status can then be displayed on the main application and if the processing state changes, then the Station Controller can be programmed to take certain actions automatically. This might include sending remote commands back to the tube to change its state. This example shows how obtaining the tube status was accomplished.

The configuration file for the furnace equipment interface looks like this:-

```
-drvname Furnace
-dbname Furnace
-sddir C:/Cell/Driver/Furnace
-devid 0
-port COM9

-start {Comms TubeStatusT1 TubeStatusT2 TubeStatusT3 TubeStatusT4}

-extend Furnace.tcl
#-debug 1
-icon
```

The MUX itself has device id 0. When the driver starts the Comms state diagram and a TubeStatus for each tube are also started and thereafter run frequently (eg. in the Tube Status case, every 5 seconds). Figure 7:State Diagram for Equipment Interface shows the Comms state diagram used at the furnace Station Controller. A TubeStatus state diagram looks like this:-



The Status S1F5 bubble does this (for tube 1):-

```
set PORT(DEVID) 1
SendMsg S1F5R {I1 0}
```

The interface manual to this system (Thermco Systems TMX 10000 Series Interface Guide) specifies the format of this message and what will be received in reply. The relevant page from it is shown here:-

4.1.2.1 Process Status #1

```
SECS Device Applicability: Tube Controllers

Process Status is a collection of variables which relate to the process state,
completion time, etc.

S1,F5 Formatted Status Request                                  S, H E, reply

Request the TMX to send Process Status for the specified diffusion tube.

Structure: <SFCD>

S1,F6 Formatted Status Data                                     S, H E

Structure:    L,2

                    <SFCD>
                    L,5
                          <PSTATE>
                          <PPID>
                          <SRECNAME>
                          <PTIME>
                          <STIME>

Define:
```

```
SFCD          Status Form Code                              Format: 10 or 31
              Length 1
              Value 0 = "Process Status #1"

PSTATE        Process State of Tube                         Format: 31
              Length 1
              Value  0 = Standby
                     1 = Processing
                     2 = Process-Standby
                     3 = Process-Initialize
                     4 = Process-Hold
                     5 = Process-Abort
                     6 = Process-Abort-Hold
                     7 = Abort
                     8 = Process Complete

PPID          Name of Current Process Recipe                Format: 20
              Length 8

SRECNAME      Name of Recipe/Subrecipe Being                Format: 20
              Executed
              Length 8

PTIME         Time left until Process                       Format: 34
              Completion in Seconds

STIME         Time left in Current Process                  Format: 34
              Delay Step in Seconds
```

The Got Status bubble does this:-

```
set TStatus [lindex [lrange $PORT(lastrmsg) 1 end] 1]

set ST(C1_tstate)  [lindex [lindex $TStatus 1] 1]
set tstate         [lindex [lindex $TStatus 1] 1]
set ST(C1_PPID)    [lindex [lindex $TStatus 2] 1]
set ST(C1_SREC)    [lindex [lindex $TStatus 3] 1]
set ST(C1_PTIME)   [lindex [lindex $TStatus 4] 1]
set PTIME          [lindex [lindex $TStatus 4] 1]
set ST(C1_STIME)   [lindex [lindex $TStatus 5] 1]

set ST(C1_remtime) [ clock format $PTIME -format "%H:%M:%S" ]
set ST(C1_estcomp) [ clock format [expr $PTIME + [clock seconds] ] -format
"%a %H:%M" ]

switch -exact -- $tstate {
      0 {set PSTATE Standby}
      1 {set PSTATE Processing}
      2 {set PSTATE Pr-Standby}
      3 {set PSTATE Initialize}
      4 {set PSTATE Proc-Hold}
      5 {set PSTATE Proc-Abort}
      6 {set PSTATE Abort-Hold}
      7 {set PSTATE Abort}
      8 {set PSTATE Complete}
}

set ST(C1_PSTATE) $PSTATE
```

The ST array provides access to the RTDB from within the driver. The C1_ prefix on the variable names in the array signifies tube 1. The variables are named as they are on the SECSII interface manual. The integer values for the tube state are translated into text values per the manual. The remaining time for the process as received from the equipment is reformatted from seconds into a more useable clock format and is also added to the current system time to produce an estimated completion time. Since furnace processes may last up to 20 hours, this estimated completion is formatted as day and time.

The effect of the above is that every 5 seconds the current status of each tube is written to the RTDB. In the main application, label widgets are created in the Production tab's constructor which will display some of the furnace tube status values as they are updated.

```
label $run.tube.state.l  -text "State" -width 10
label $run.tube.state.r  -textvariable [statusView attachVar \
${dbPrefix}PSTATE] -width 12 -bg white -fg black
pack $run.tube.state.l -side left -padx 10
pack $run.tube.state.r -side left -padx 10
label $run.tube.recipe.l  -text "Recipe" -width 10
label $run.tube.recipe.r  -textvariable [statusView attachVar \
${dbPrefix}PPID] -width 12 -bg white -fg black
pack $run.tube.recipe.l -side left -padx 10
pack $run.tube.recipe.r -side left -padx 10
label $run.tube.subrcp.l  -text "Subrecipe" -width 10
label $run.tube.subrcp.r  -textvariable [statusView attachVar \
${dbPrefix}SREC] -width 12 -bg white -fg black
pack $run.tube.subrcp.l -side left -padx 10
```

```
pack $run.tube.subrcp.r -side left -padx 10
label $run.tube.time.l  -text "Time left" -width 10
label $run.tube.time.r  -textvariable [statusView attachVar \
${dbPrefix}remtime] -width 12 -bg white  -fg black
pack $run.tube.time.l -side left -padx 10
pack $run.tube.time.r -side left -padx 10
label $run.tube.estc.l  -text "Est. Comp." -width 10
label $run.tube.estc.r  -textvariable [statusView attachVar \
${dbPrefix}estcomp] -width 12 -bg white  -fg black
pack $run.tube.estc.l -side left -padx 10
pack $run.tube.estc.r -side left -padx 10
```

Since the main application code creates objects, the RTDB prefix (from the configuration file) for this particular object will be appended to the variables when it is created. Thus the prefix for tube 1 is C1_ and so the values for tube 1 will be displayed in the tab for tube 1 (and not somewhere else, which might confuse people!).

The actual messages being sent & received can be viewed via one of the interface applications and will look like this for 1 inquiry for tube 3 only:-

```
{{2002-05-23 14:54:28} S1F5R {I1 0} {State_Machine_Reply_Callback
TubeStatusT3 1 4 {

}}}
{{2002-05-23 14:54:28} S1F6 {L:2 {I1:1 0} {L:5 {I1:1 1} {A:8 START192} {A:8
START192} {I4:1 16334} {I4:1 16334}}} 82956 {after#248863
{State_Machine_SECS_Callback TubeStatusT3 1 5 {
```

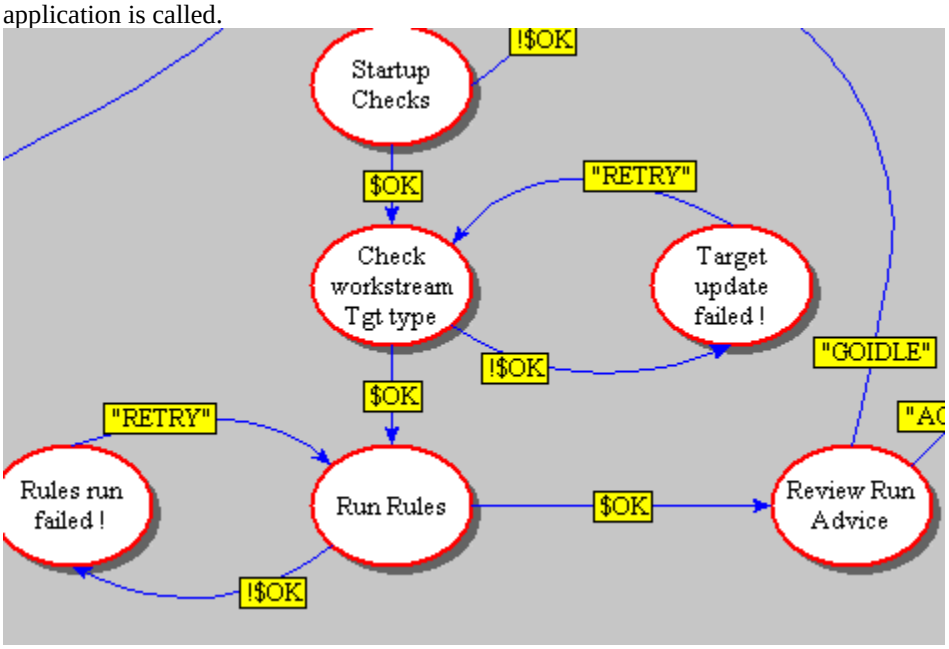
```
} - {}} 1 5 {State_Machine_Reply_Callback TubeStatusT3 1 4 {

}}} {}}
```

Complex Rules to Control a Sputter Reactor

A Sputter Reactor is a high vacuum system that coats silicon wafers with thin films of metals from solid targets of the desired metal or alloy. When a Station Controller for Sputter Reactors was under development, the process engineers requested that the software control not just a single run but the entire running of the machine. The process specification would, in effect, be translated into the Tcl code in the Station Controller. The history of recent activities would be kept in the RTDB. For example, certain kinds of preventative maintenance actions might dictate that special test runs be performed before running production. At other times, changing sputter targets might necessitate different measurements being performed or if a target had not been used for some time, then special conditioning might need to be carried out before it could be used again.

The State Diagram concept was used and extended to cover this situation. Although, code could have been written to meet the requirements without using one, it was realised that using a State Diagram to control the logical flow would make it much clearer and easier to understand. The same control could have been achieved by using long, complex if { …} {…..} elseif {…} {….} elseif {…...} code. However, this would have been difficult to manage and modify. The State Diagram is easy to manipulate and the logical flow is crystal clear even to non-programmers. This Rules Engine State Diagram is separate from the main one used to control the Station Controller. Here is how this works.

When a run is initiated, the Start Checks are performed as described above but then, a runRules method in the main application is called.



This method calls methods in a separate object specifically created to define the required rules:-

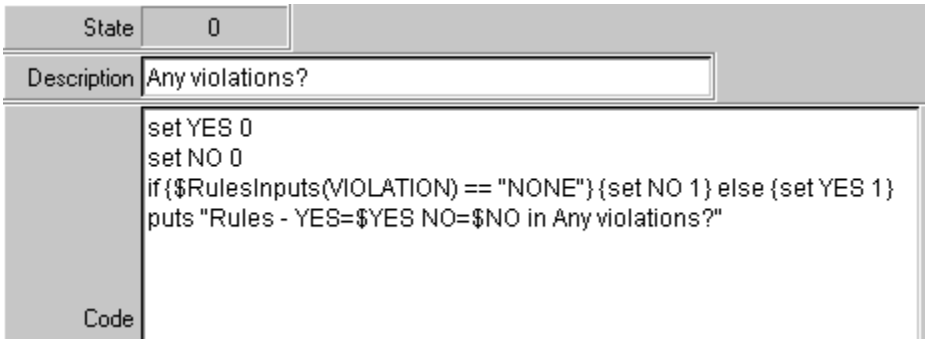
```
method runRules {} {
            global _TDL_ etchrecipe TARGETS
            if {$etchrecipe != "NONE"} {
                  set RECIPE $etchrecipe
```

```
            }
            if [catch {runadviceMgr runRules} msg] {
                  Log_write ERROR "Rules run Failed $msg"
                  return -1
            }
.
.
}
```

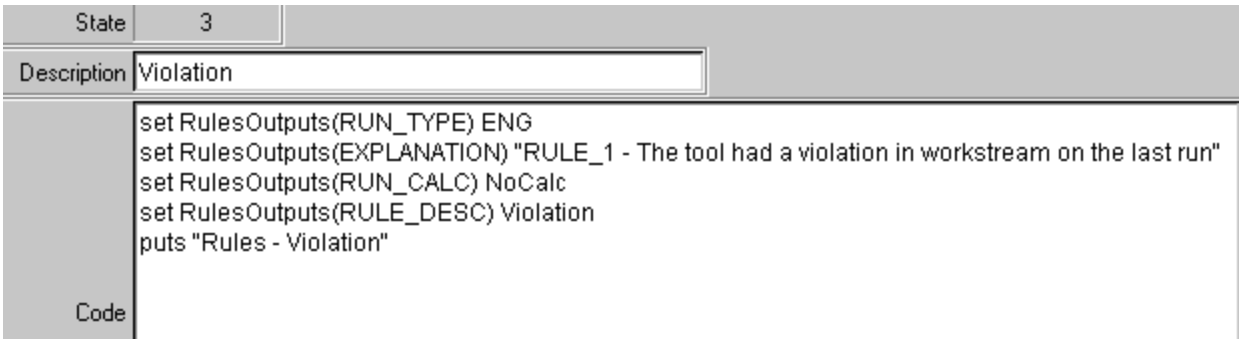
The runadviceMgr object's runRules method will typically execute some leg of the rules State Diagram:-

```
method runRules {} {
            clearValues
            writeRulesInputsArray
            Log_write DEBUG "Run rules starts"
            if {$prev_sheet_res == "" || $prev_deptime == "" || \
                  $last_run_predn_dep_rate == ""} {
                        rtdbIF writeone ${dbPrefix}RUN_TYPE "ENG"
                  set RECOMMENDED_RUN_TYPE $RUN_TYPE
                        set explanation  "Rules not run - check the last run
\
                              data for this target"
                  rtdbIF writeone ${dbPrefix}EXPLANATION $explanation
                  set DISP_run_calc       "NoCalc"
                  set DISP_target         [rtdbIF readone \
                        ${dbPrefix}TARGET_IN_USE]
                  return
            } else {
                  runRulesSD
                  readRulesOutputsArray
                  productionMgr reset_run_display
            }
      }
```

A portion (approximately the top 1/3) of the Sputter Rules State Diagram is shown in Figure 10. A sequence of methods will be called, each of which will test some condition based on data stored in the RTDB. The result of the test is a YES or NO answer to the question posed in the bubble's name label.



To illustrate this, the first bubble encountered asks "Any Violations?":-
A particular element of a rules input array (created when the rules object was created and modified through usage of the Station Controller) is examined and the State Diagram branches depending on the value in the array. If the answer is NO, the next question in the diagram will be answered. If the answer is YES (there are violations), the next bubble creates an output array, parts of which will be displayed back to the operator as Run Advice.



The operator either accepts the advice, in which case the particular run type recommended will be started, or does not, in which case the Station Controller returns to its initial Idle state and no further processing can be carried out.

Using the WorkStream Interface

The relatively simple example of obtaining an entity status from WorkStream is described here. This might occur at several points within the main application and, in particular, will be used when the Start Checks method described in Typical Use is called. Refer to Figure 9:MES Interface Architecture for an overview of what happens.

When the Start button on a Station Controller is pressed by the operator, a method to initiate various checks is performed. These will vary from Controller-type to Controller-type but will all look something like this:-

```
method startChecks {} {
            # Make sure our idea of the Entity Status is up to date

            statusMgr refresh
            set status [statusMgr status 1 3 6 9]
            set status1 [lindex $status 0]
            set status3 [lindex $status 1]
            set status6 [lindex $status 2]
            set status9 [lindex $status 3]

            if {$LOT_NUMBER == ""} {
                  Log_write ERROR "Cannot start - No Lot Selected"
                  return -1
            }

            if {$status1 != "PRODUCTION"} {
                  Log_write ERROR "Cannot start - $entity not in Production"
                  return -1
            }
            if {$status3 != "STANDBY" && $status3 != "NO WORK" && \
            $status3 !=  "NO OPERATOR"} {
                  Log_write ERROR "Cannot start - $entity not in Standby"
                  return -1
            }
.
.
.
}
```

This uses a method (refresh) already defined in the standard Status tab to inquire through the MES interface about the entity status. An entity in WorkStream is its representation of a piece of equipment and is defined for the Station Controller in the main application's configuration file. The initial part of this standard library code method looks like:-

```
method refresh { {data ""} } {

            # Get Data from Workstream if data is not passed in.
            if {$data == ""} {
                  if [catch {mesIF getEntityInfo $entity} data] {
                        Log_write DEBUG "getEntityInfo <$entity> FAILED - \
                         $data"
                        return -1
                  }
            }
```

This calls another library method (getEntityInfo) defined in the httphostif code which is still on each Station Controller. Here is the relevant excerpt:-

```
method getEntityInfo {entity} {
            # Log_write DEBUG "Get_entity_info ($entity) ..."
            if [catch {toMES ENST FACILITY $facility USERNAME $USERNAME \
            PASSWORD $PASSWORD ENTITY $entity} msg] {
                  Log_write EXCEPTION [tidy $msg]
            }
            # Log_write DEBUG "Get_entity_info ($entity) Successful - $msg"

            set result [list]
            foreach {name value} [lrange $msg 10 end] {
                  lappend result $value
            }
            return $result
}
```

The WorkStream function code to obtain an entity status is ENST. The facility variable is defined in the Controller's configuration file and username & password and supplied by whoever logged into the Station Controller. The toMES method sends the request to one of the Linux servers where a cgi script called ENST is executed. Most of these cgi scripts are nearly identical other than defining the tcl function to be called and the variable values passed to it. In this case, this is what is in the cgi script:-

```
#!/usr/bin/tclsh8.3
source /home/cim/mesif/msglib.tcl
package require ncgi
ncgi::parse
ncgi::importAll

set box [pid]
clear_mbox $box
put_msg WS [list $box ENST $FACILITY $USERNAME $PASSWORD $ENTITY]
set resp [get_msg $box 45]
delete_mbox $box

puts "Content-type: text/plain\n"
set st [lindex $resp 0]
puts [list Result $st]

if {$st == "***SUCCESS***"} {
        foreach {name value} [lrange $resp 1 end] {
                puts [list $name $value]
        }
} else {
        puts [list Message [lrange $resp 1 end]]
}
exit 0
```

To understand what happens next, here are the WorkStream screens that the Expect scripts will emulate:-

```
NTC220Sf                  RTC INQUIRIES MENU (NIQM)       SYS  5/23/02 15:24:18
UKDB02 V5.5-002 PROD               UK4106



                              FUNCTION...ENST
            ENTITY.....ALIGN-8_____     EVENT SCHEME NUMBER....._
(STARTING) EVENT......_____________     EVENT GROUP VALUE (opt)._____________
            CUTOFF DATE....__/__/__

                RTC ENTITY STATUS.........................(ENST)
                RTC VIEW ENTITY ALARM HISTORY.............(VNAL)
                RTC VIEW AVAILABLE ENTITIES FOR AN EVENT..(VAES)
                RTC VIEW COMING EVENTS FOR AN ENTITY......(VCVE)
                RTC VIEW SCHEDULED EVENTS FOR AN ENTITY...(VSEN)
                RTC VIEW ENTITY LOT LIST..................(VELL)
                RTC VIEW EVENTS BY A GROUP................(VVBG)
                RTC VIEW SELECTED CLOCKING ACTIVITY.......(VSCA)

RETURN = PROCESS
SFK1   = EXIT
```

```
NTC221S                   RTC ENTITY STATUS (ENST)      SYS  5/23/02 15:27:07
UKDB02 V5.5-002 PROD               UK4106

                       ENTITY.............ALIGN-8
    ENTITY TYPE........ALIGNER             RTC MODEL.......
                       DESCRIPTION........ATEGO ALIGNER NO. 8
                       ENTITY DELETED.....NO

     STATUS CATEGORY       VALUE             STATUS CATEGORY       VALUE
     ---------------     ------------        ---------------     ------------
 S1  RESPONSIBILITY      PRODUCTION      S6  MONITOR STATUS      OKAY
 S2  SUB-SEMI E10        RUN PROD        S7  1st MASK COUNT      0
 S3  DETAILED STATE      RUNNING PROD    S8  FAILURE CODE        NONE
 S4  MAINT. TECH         NO ONE          S9  CHECKS OVERDUE      NONE
 S5  RESTR               RESTR
                          AVAILABILITY.......UP

  LAST EVENT.........ALIGN-START              TRANSACTION DATE... 5/23/02
                                              TRANSACTION TIME...15:06:49

SFK1 = EXIT              SFK2 = HELP              SFK3 = NEXT FUNCTION..
```

The ENST.tcl script goes to the screen with form name (top left corner) NTC220Sf, puts the value of entity into the first field and sends a Return. It then expects to see the NTC221S screen appear or to still be at the first screen and have detected an error. The error text will be returned to the Station Controller. If no error was encountered, the NTC221S screen is parsed by row, column start position and field length. The fields so extracted are built into a results list that is returned to the Station Controller. The Expect session is then returned to a login (LOGN) screen to await the next request from another Station Controller.

Here is the code that performs this:-

```
proc ENST {facility user password entity} {
        global term

        prepare_form $facility $user $password ENST NTC220Sf

        exp_send [format "%-12s\r" $entity]
        term_expect {expr {[curscreen] == "NTC221S" && [cursor@ 21 75] }} {} \
                    "expr {\[curscreen\] == \"NTC220Sf\" && \[bell_sounded] && \
             \[term_match 22 0 22 78 ERROR\] \
       }" {flag_error} \
                    {timeout} {flag_timeout}

        check_errors

        set field_defs { EntityType     5 24 19         RTCModel        5 59 20
                         Description    6 43 35         EntityDeleted   7 43 3
                         StatusCat1     11 5 15         StatusVal1      11 25 12
                         StatusCat2     12 5 15         StatusVal2      12 25 12
                         StatusCat3     13 5 15         StatusVal3      13 25 12
                         StatusCat4     14 5 15         StatusVal4      14 25 12
                         StatusCat5     15 5 15         StatusVal5      15 25 12
                         StatusCat6     11 44 15        StatusVal6      11 64 12
                         StatusCat7     12 44 15        StatusVal7      12 64 12
                         StatusCat8     13 44 15        StatusVal8      13 64 12
                         StatusCat9     14 44 15        StatusVal9      14 64 12
                         StatusCat0     16 26 12        StatusVal0      16 45 5
        }

        update
        set results [build_results $field_defs]
        # We can use KP_3 instead of find_home here.
        exp_send "LOGN[KP_3]"
        term_expect {expr {[curscreen] == "CMT009S"}} {}
        find_home
        return $results
}
```

Back in the main application, this results list is then parsed for the important information in the startChecks method using a standard method from the statusMgr object.

Benefits to Manufacturing

Commercial confidentiality considerations prevent exact or specific details being given here.

Using Station Controllers at some processes has lead to throughput being improved to much higher levels than were previously possible and these improvements have been sustained. Higher throughput implies shorter cycle times and greater capacity for costs significantly lower than purchasing additional fab equipment. This throughput improvement was achieved because the Station Controller imposed rigorous, consistent processing that did not rely on operators's memories and eliminated unnecessary test runs or equipment conditioning thus allowing more production wafers to be run.

As outlined in the introduction, a major benefit of Station Controllers is the use of recipe download. This prevents incorrect recipe selection and improves yield. In some cases, 'wrong recipe' has been eliminated as a reason for scrap. Indeed, as the major causes of scrap are removed, the processes that formerly caused relatively insignificant amounts become the top cause. This often drives the priority for the next implementation of a new Station Controller type.

Computing equipment becomes obsolescent at an alarming rate whereas fab processing equipment vendors still support equipment that might be 10-15 years old. However, if a PC system was included with a piece of kit at original purchase, it is now likely to be impossible to support. Constructing a Station Controller for such equipment allows the obsolete system to be replaced with a modern PC and operating system. At the same time, recipe download can be introduced to improve yield.

A similar approach has been taken to replace some legacy download applications that used obsolescent software (eg. DEC MessageQ or Brook's Grapheq) that are also difficult to support since knowledge and understanding of old packages like these has vanished. New features or requirements can be incorporated into a Station Controller that would have been difficult or impossible before.

Figure 1:

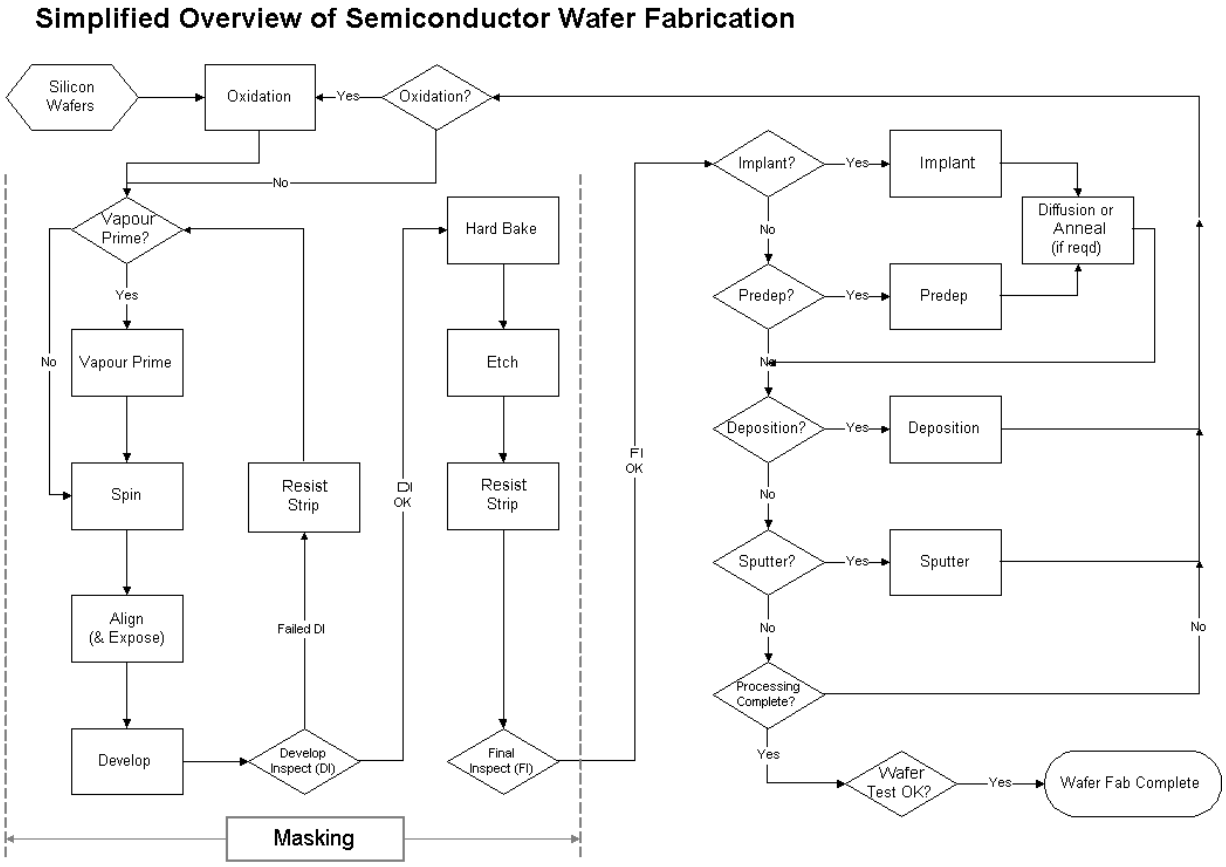

Simplified Overview of Semiconductor Wafer Fabrication
Silicon Wafers
Oxidation
Oxidation?
Yes
No
Vapour Prime?
Yes
No
Vapour Prime
Spin
Align (& Expose)
Develop
Resist Strip
Failed DI
Develop Inspect (DI)
DI OK
Hard Bake
Etch
Resist Strip
Final Inspect (FI)
FI OK
Implant?
Yes
Implant
No
Diffusion or Anneal (if reqd)
Predep?
Yes
Predep
No
Deposition?
Yes
Deposition
No
Sputter?
Yes
Sputter
No
Processing Complete?
No
Yes
Wafer Test OK?
Yes
Wafer Fab Complete
Masking

Figure 2: Station Controller Concept

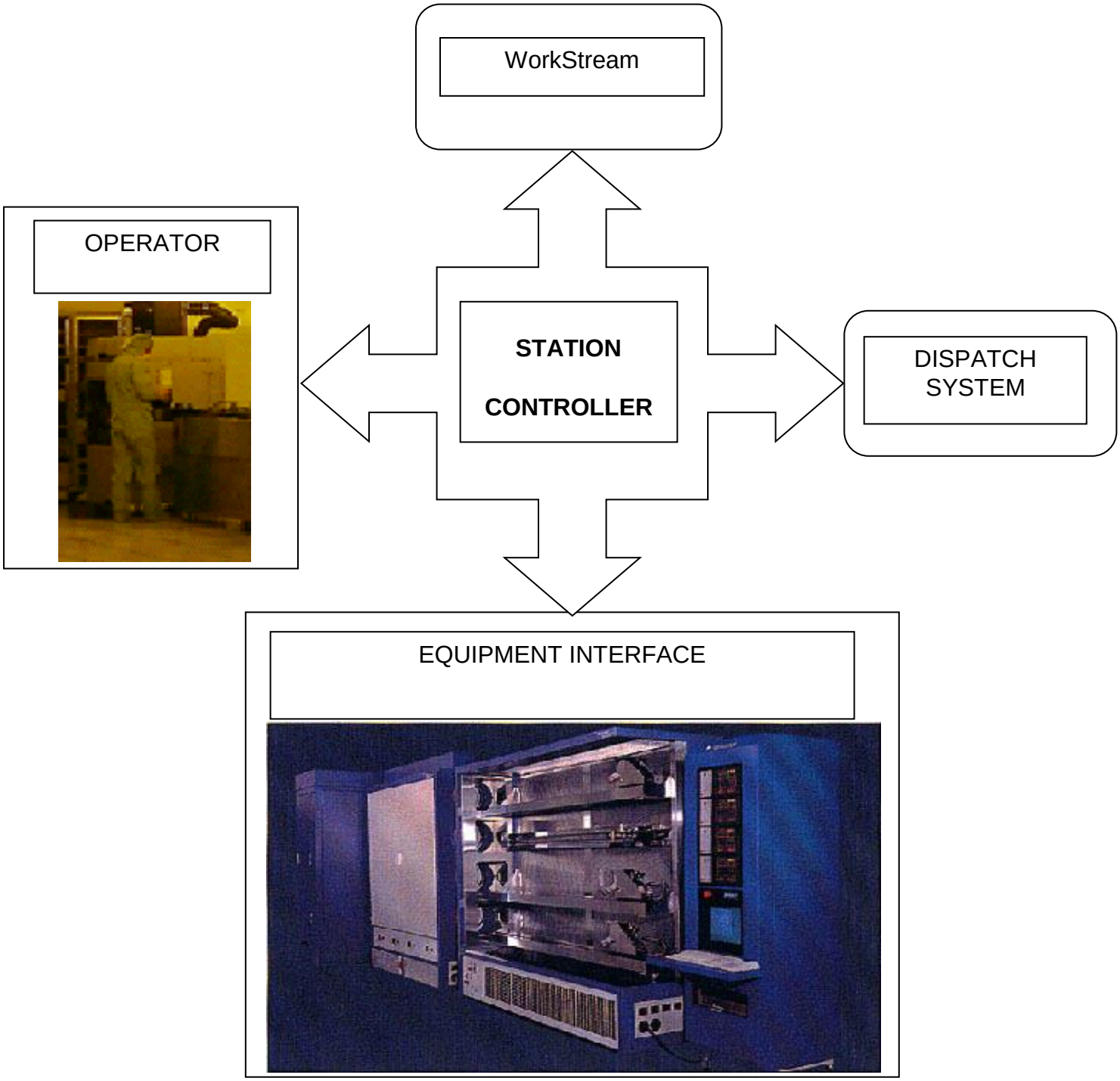


Figure 3: A typical Station Controller Screen

Figure 4: An Example of a State Diagram

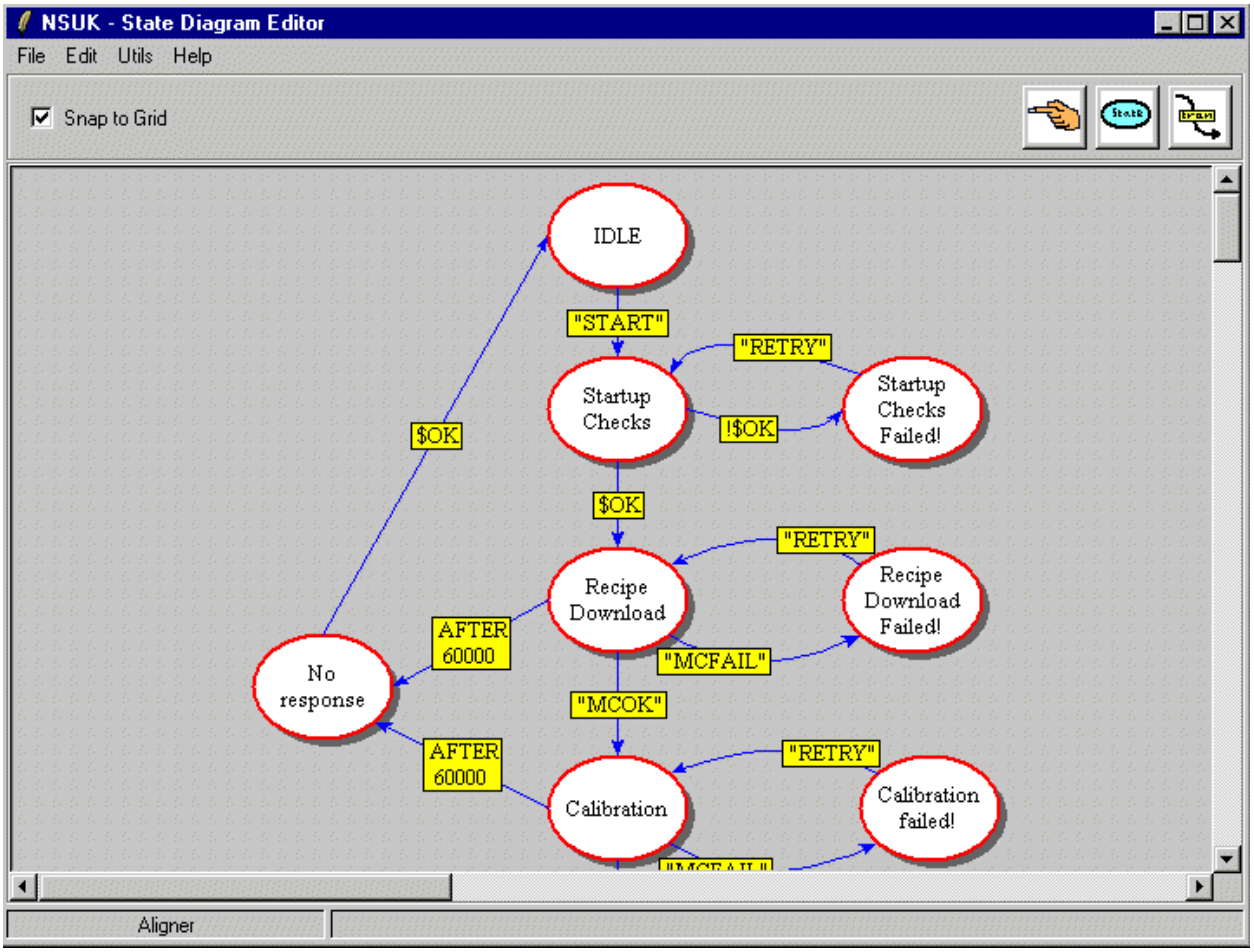


Figure 5: Updating Code in a State Diagram

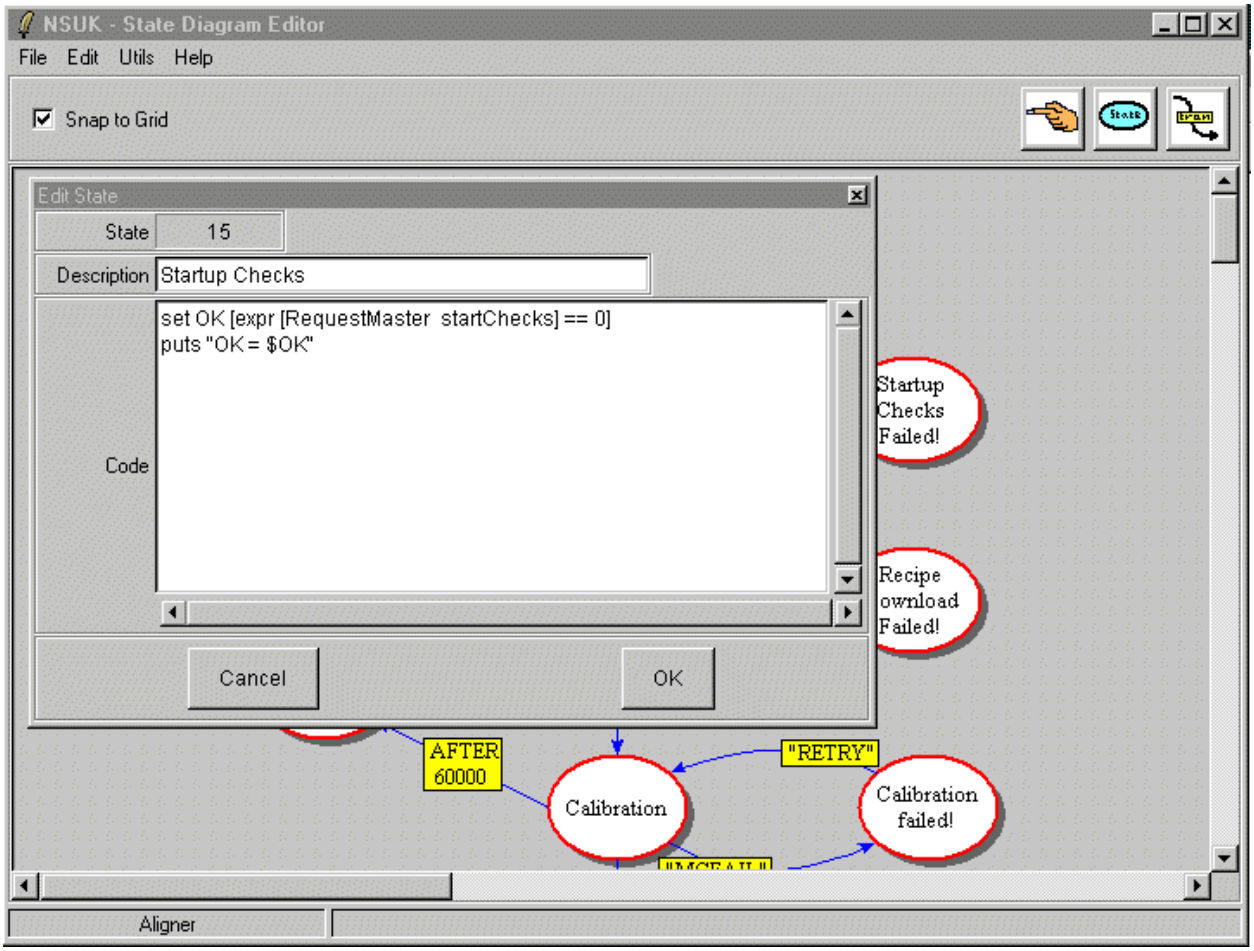


Figure 6: Extract from State Diagram file

```
{15 {Startup Checks} {set OK [expr [RequestMaster  startChecks] == 0]
puts "OK = $OK"



} {
      {{EXPR !$OK} {
} {16} {R L 459.0 162.0}}
      {{EXPR $OK} {
} {38} {B T}}
      } {350.0 140.0}
}
{16 {Startup Checks Failed!} {

}
```

Figure 7: State Diagram for Equipment Interface

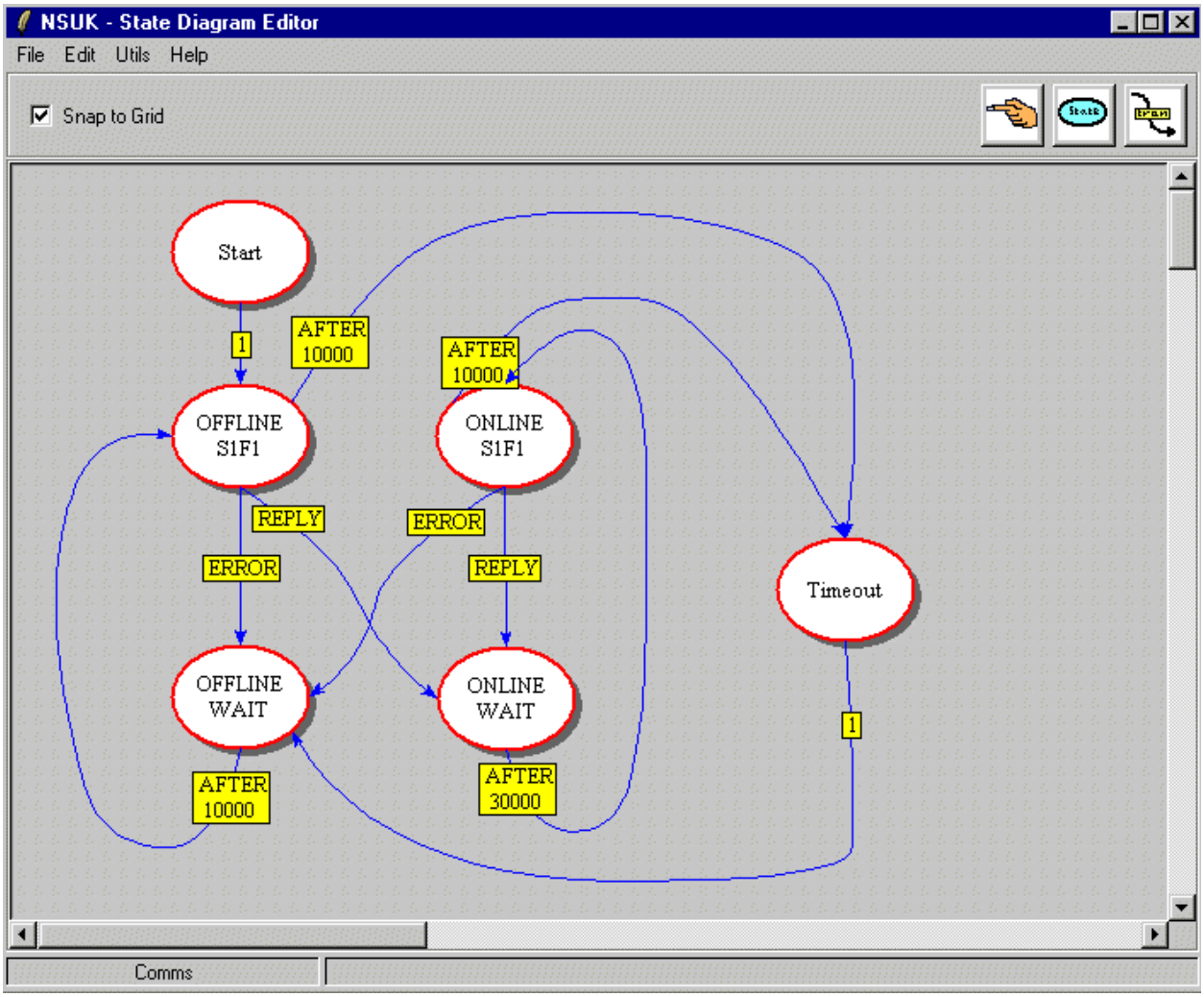


Figure 8: Implant RecipeSetup State Diagram

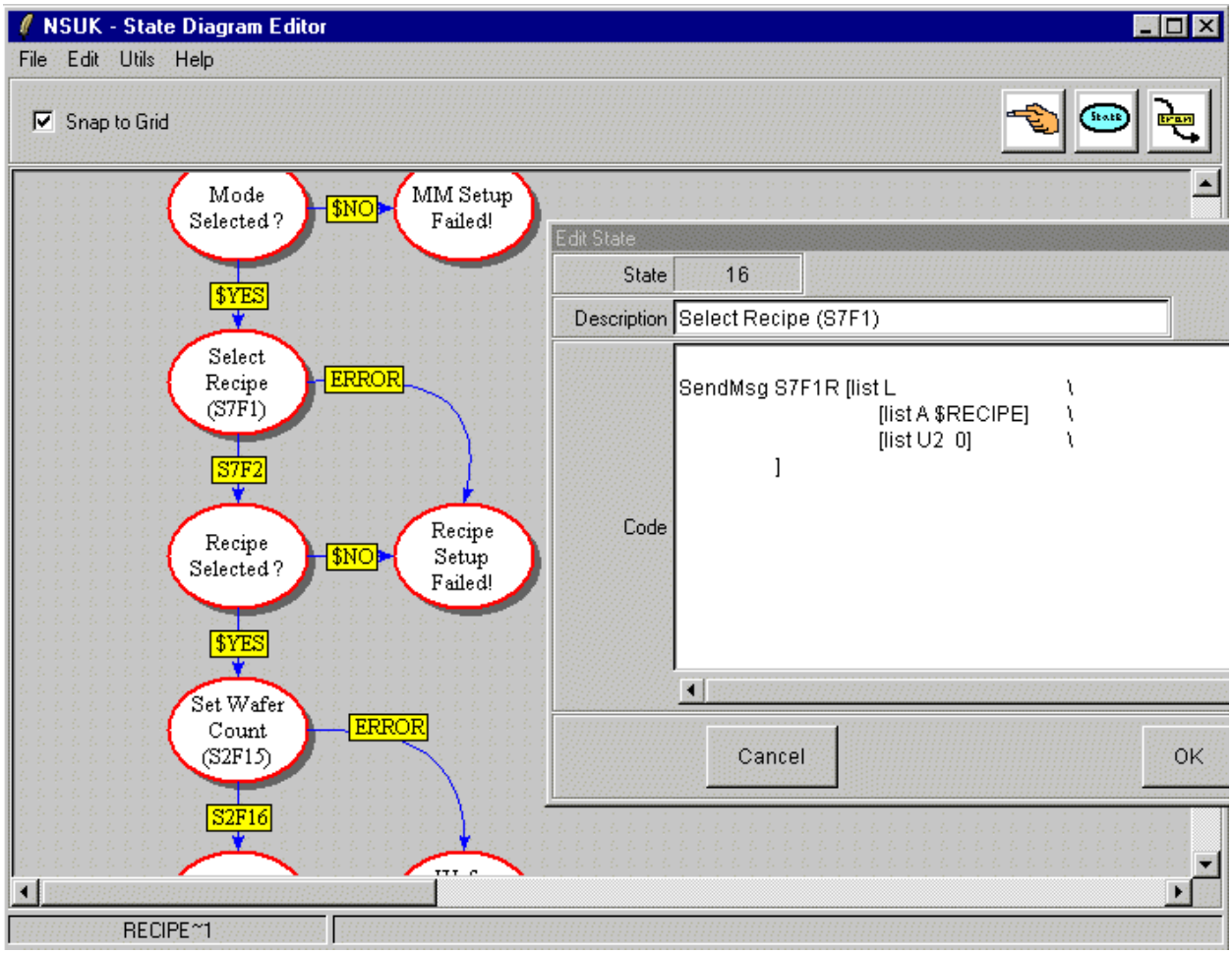


Figure 9: MES Interface Architecture

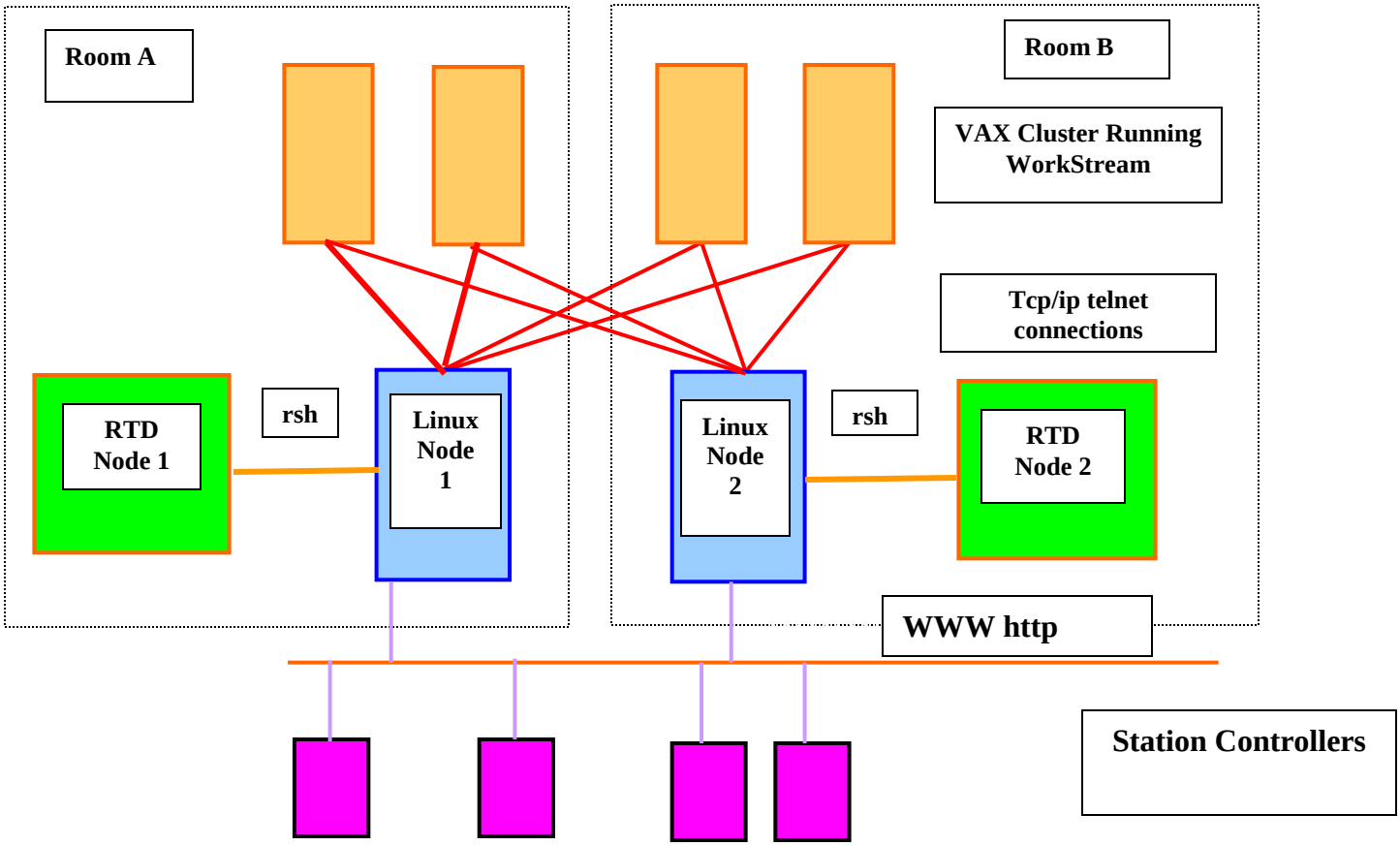
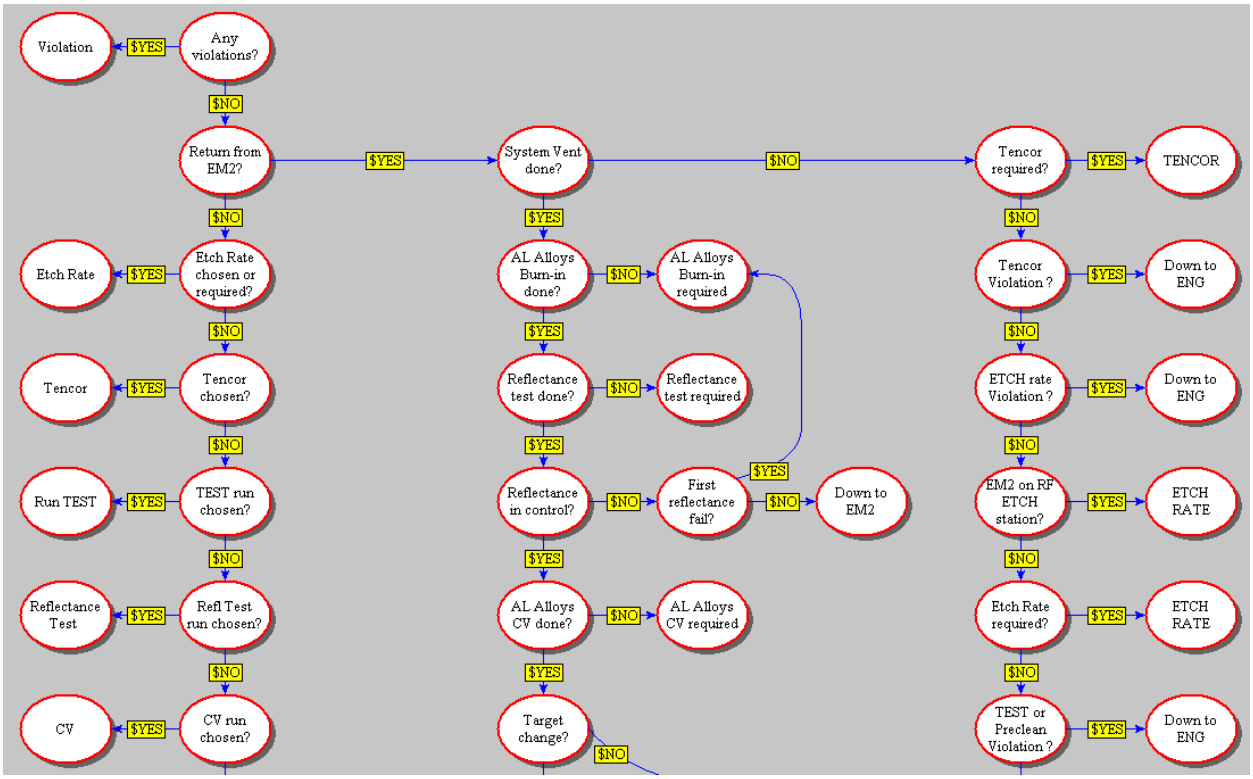


Figure 10: A portion of the Sputter Rules State Diagram



Biographical Details

Mr. Campbell Boyd graduated in 1979 with a BSc. (Hons) in Physics from the University of Glasgow. He worked at National Semiconductor (UK) Ltd. as a process engineer for 5 years and then as a systems engineer, latterly implementing WorkStream in the role of key user. In 1986, he joined Siemens AG at their Munich, Germany semiconductor division and continued to work with WorkStream and data analysis. He returned to National in 1989 and developed technical data capture and analysis systems using VAX & Sun platforms. Between 1996 and 1999, he undertook project and programme management roles for yield improvement. From 2000 until the present, he has returned to a technical role developing and supporting factory automation systems. He is currently a Principal Factory Systems Engineer in the Systems Development group within the IS Department at National Semiconductor's Greenock, Scotland facility.

Contact

Campbell Boyd

Information Systems Department
National Semiconductor (UK) Ltd.
Larkfield Industrial Estate
Earnhill Road
Greenock
Scotland
PA16 0EQ

Telephone +44 (0)1475 655301
Fax +44 (0) 1475 637755

Email: Campbell.Boyd@nsc.com