GrafTab — An Innovative Requirements Specification Method

— A Requirements Specification of a Steam-Boiler Controller —

So-Ming So, Mabo R. Ito
Department of Electrical Engineering
The University of British Columbia
B.C. Canada

October 6, 1995

1 Introduction

An innovative requirements specification method, GrafTab, is introduced in this paper. The main characteristics of GrafTab are that graphs are used to describe the flow of the system states and tables are used to specify the state condition and actions in the states. The primary goal of developing this new requirements specification method is to specify a system in a complete, correct, and readable way. Another goal is that the method should not be too difficult and complicated to learn. In order to accomplish these goals, four distinctive documents are used in *GrafTab*. They are the system abstract, the monitored and controlled variables specification, the external interface document, and the system requirements document. The rationale behind *GrafTab* is that a system first is described briefly by the system abstract. The environmental variables are then identified in the monitored and controlled variables specification and the external interface document. Finally, the flow of the system states and the relationship among the environmental variables in the states are specified in the system requirements document. In the latter, the flow of the system states is described by an Enhanced State Diagram (ESD), which is adopted from Statechart [2]. The relationship among the environmental variables in each primitive state is specified by a State Specification (S-Spec) consisting of two functional tables, namely State Action Table (SAT) and Control Action Table (CAT).

A requirements specification of a steam-boiler controller [1]will serve as a running example of *GrafTab*.

2 GrafTab

The four documents in *GrafTab* are defined formally in the following sections. The syntax and properties of the ESD and the S-Spec are described as well.

2.1 System Abstract

Regardless of the size of a control system, a basic understanding of the system, such as the overall function of the system and the type of data input and output, is very important. The system abstract is to be written in English, the most accessible and readable language to most readers. When readers have any questions about the system, they must always refer to the other three documents for the formal description of the system.

Even though no specific guideline will be provided in writing the system abstract, the abstract must be precise and concise. A long abstract will normally provide unnecessary and ambiguous information about the system, which adds burden to readers.

The following is the system abstract for running example: steam-boiler controller.

System Abstract: Steam-boiler Controller

The system comprises of the following physical units: a steam-boiler, a water level measuring unit, four pumps to provide the steam-boiler with water, four pump controllers, and a steam outcome measuring unit.

The program receives signals from the physical units, analyzes the signals, and then sends back the responses to the physical units in every five seconds.

The program is started by receiving the signal **Steam_Boiler_Waiting** from the physical units. The program checks if the quantity of steam coming out of the steam-boiler is zero. If it is not zero, the program will stop. The program also tries to maintain the water level in the steam-boiler between N_1 and N_2 by opening the valve of the steam-boiler or activating the pumps. If the water level measuring unit is defective, the program will stop. As soon as the water level is between N_1 and N_2, the program sends continuously the signal **Program_Ready** to the physical units until it receives the acknowledgment signal from the physical units. Then, if all the physical units operate correctly, the program will enter the normal mode. If any physical unit is defective, the program will enter the degraded mode.

In the normal mode, the program tries to maintain the water level in the steam-boiler between N_1 and N_2 by switching the pumps on or off. Except for failure in the water level measuring unit, all physical unit failures put the program into the degraded mode. In the degraded mode, it is assumed that the water level measuring unit works correctly. When all defected physical units are repaired, the program goes back to the normal mode. In both the normal mode and the degraded mode, if the water level measuring unit is defective, the program will enter the rescue mode. Also, when the water level reaches one of the limit values M_1 and M_2, the program will stop.

The water level in the rescue mode is estimated from the maximum dynamics of the quantity steam coming out of the steam-boiler. As soon as the water level measuring unit is repaired, the program returns to the normal mode or the degraded mode. The program stops if one of the following cases holds: the steam outcome measuring unit is defective, one of the pump controllers has a failure, or the water level reaches one of the limit values M_1 or M_2.

Since this is an informal document, readers must consult the other three documents for the formal specification of the controller.

2.2 Monitored and Controlled Variables Specification

This document is used to specify the inputs and the outputs of a system. It consists of the monitored variables section and the controlled variables section.

2.2.1 Monitored Variables The inputs of a system are designated as the monitored variables in the *GrafTab* specification method because they are always monitored by the system. In this document, each monitored variable will be given a name and a set of possible values. However, no initial value for the monitored variables are to be specified, as the system specification document is able to describe the system correctly with any given input state. The following is one of the monitored variables in the steam-boiler controller. In this example, "Level" is the name of the variable. The second line "quantity of water in the steam-boiler" explains the variable. The last line defines the possible values of the variable.

Monitored Variables:

Level

— quantity of water in the steam-boiler

— possible values: Real (liter)

2.2.2 Controlled Variables The outputs of a system are designated as the controlled variables in *GrafTab* because they are usually used to control some external or internal terminals. Each controlled variable will be given a name and a set of possible values. Since the program always has control over the outputs, the initial value of each variable is stated as well. If the controlled variable is also used as an input of the system, it is still considered to be a controlled variable, as the input can be referenced internally in the system. The following is one of the controlled variables in the steam-boiler controller. It has the same format as the monitored variable except that the initial value is specified. See Appendix A for a complete monitored and controlled variables specification of the steam-boiler controller.

Controlled Variables:

Valve

- — valve for evacuation of water from the steam-boiler
- — possible values: {Open, Close}
- — initial value: Close

2.3 External Interface Document

In this document, the physical addresses of the monitored variables and the controlled variables are specified. Except for the internal monitored and controlled variables, each variable must be mapped onto a unique physical address. This document is divided into two sections: input addresses and output addresses.

2.3.1 Input Addresses This section is used to specify the physical addresses of the external monitored variables. Each of the external monitored variables defined in Section 2.2.1 must be mapped onto a physical input address. The type of input is also specified such as DC input or analog input.

2.3.2 Output Addresses The output addresses are the physical addresses of the external controlled variables defined in Section 2.2.2. Each external controlled variable is mapped onto one physical output address. The type of output is also specified.

The external interface document of the steam-boiler controller is not included in this paper because the physical interface between the system and the physical units is unknown at this stage.

2.4 System Requirements Document

The system requirements document is the most important document in the *GrafTab* specification. It describes the functionality of a system thoroughly and formally. The document first identifies the system states, using an Enhanced State Diagram (ESD), and then specifies the relationship among monitored and controlled variables in each primitive state with a State Specification (S-Spec).

2.4.1 Enhanced State Diagram (ESD) The Enhanced State Diagram (ESD) is similar to a conventional state diagram in the sense that both use graphical descriptions to show the control flow in a system. The conventional state diagram, however, is a flat diagram which shows all states on one level. Thus, the diagram becomes increasingly complicated and unclear as the number of possible transitions and states increases. It is also very difficult to present a parallel action in a conventional state diagram. In contrast, the ESD has a hierarchical structure in which the top-most level provides an outline of the system states and the lower levels contain more details of the system. The mechanism for specifying parallel actions and broadcast transition are also presented in the ESD. The following subsections describe the syntax and properties of the ESD.

2.4.1.1 State A state is represented by a rounded rectangle in the ESD. There are two general and two special states in the ESD. The general states are a super-state and a sub-state; the special states are an entry state and a concurrent state. A state is activated if its predecessor state has already been activated and the transition condition of the entering transition is true. Right before the state is activated, its predecessor state is deactivated. Once the state is activated, it is kept active until one of its exiting transition condition becomes true. The transition and the transition condition are described in Section 2.4.1.2.

Super-State and Sub-State The ESD has a hierarchical structure in that it consists of different levels of states. The top level state is called a super-state of the states beneath it; the states beneath a super-state are called sub-states. A super-state must have more than one sub-state if it is to exist. A sub-state can have its own sub-states; thus, it becomes a super-state. In other words, a state can be a super-state, provided that there are some sub-states below that state in the hierarchical structure. If a sub-state does not have any sub-states, it is a primitive state and must be specified by a S-Spec (see Section 2.4.2).

Each super-state has a super-state name and a super-state number on the top left-hand corner of the rounded rectangle. Each sub-state has a sub-state name and a sub-state number inside the rounded rectangle (Figure 1). A hierarchical structure is inherent in a super-state number. For example, *0.1.2* tells you that this super-state is at level 2 of the ESD because there are two dots in the number; *.2* is the sub-state number at level 1; *.1* is the sub-state number at level 0. Note that level 0 is the top-most level in the ESD.

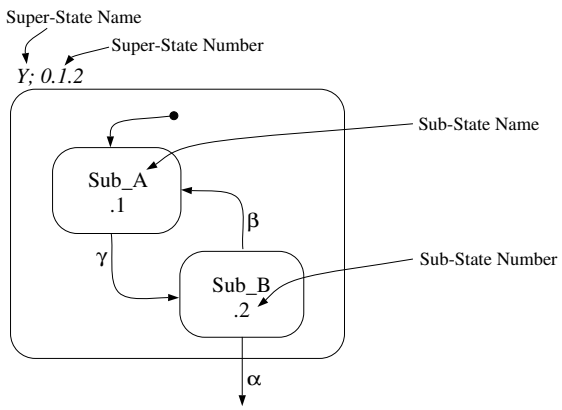


Figure 1: Super-State and Sub-State

Running example: Steam-boiler Controller There are many different ways to define the system states in the steam-boiler controller. In general, the states can be grouped into two super-states: i) states related to the steam-boiler operations ("Steam-boiler" state in Figure 2), and ii) states related to the emergency stop ("Emergency" state in Figure 2). When the controller receives the signal Steam_Boiler_Waiting, it enters the steam-boiler operations state. If an error happens in the steam-boiler operations state, or an emergency stop signal, "Stop = True", is received by the controller, the controller enters the emergency stop state.

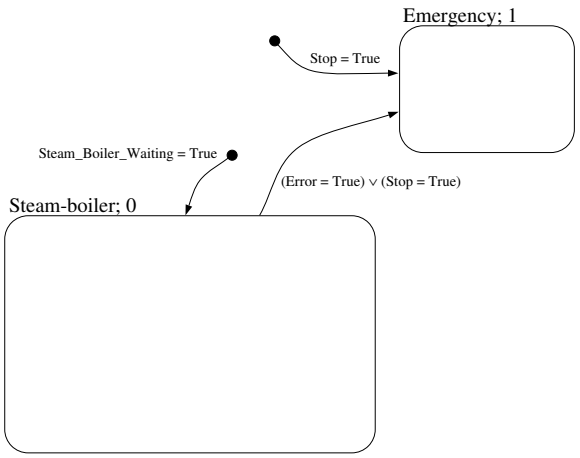


Figure 2: Level 0 ESD of the steam-boiler controller

Entry State An entry state is the starting sub-state of a super-state. When a super-state is activated by a transition which points to the boundary of the super-state, the entry state will always be activated first. The entry state is designated by an entry transition, which is an open-ended transition inside the super-state (Figure 3). There can be only one entry state and one entry transition within a super-state because only one sub-state can be activated when the super-state is active.

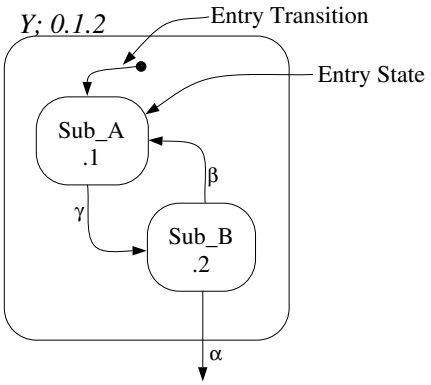


Figure 3: Entry State

Concurrent State A rounded rectangle, divided into components with dashed lines, represents a concurrent state (Figure 4) in the ESD. When a concurrent state is activated, the entry states in each component will be activated simultaneously. For example, in Figure 4, when concurrent state Y is activated by the transition with the condition γ, the system enters the sub-states A and B at the same time. If the transition condition α becomes true, the system is transformed into the sub-states A and C. Note that we can only exit super-state Y from its sub-state C, if and only if the transition condition δ is true and sub-state C is activated.

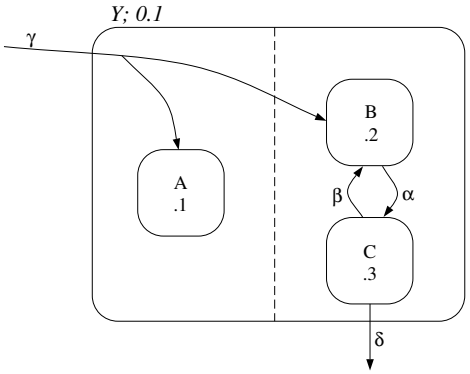


Figure 4: Concurrent State

Running Example: Steam-boiler Controller In the "Steam-boiler" state in Figure 2, the controller needs to (a) monitor the sampling time, (b) send appropriate signals to the physical units, and (c) check the physical units simultaneously. These parallel actions are easily represented by a concurrent state in ESD. Figure 5 shows a "Steam-boiler" concurrent state which contains three components: a "Sample Timer" sub-state, a "Boiler Operations" sub-state, and an "Error Checking" sub-state.

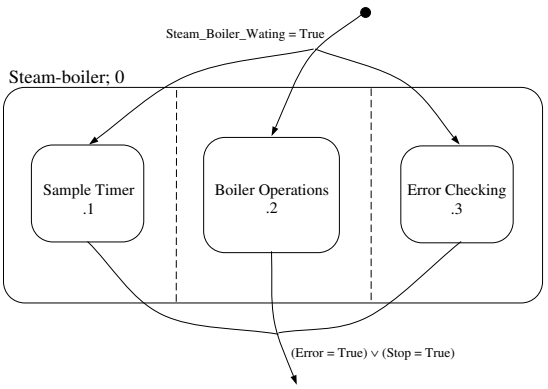


Figure 5: Steam-boiler concurrent state

Common State A common state is formed if there is more than one transition entering the state (Figure 6). The transitions, however, must be mutually exclusive, meaning that only one transition can be enabled at any time.

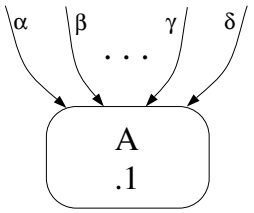
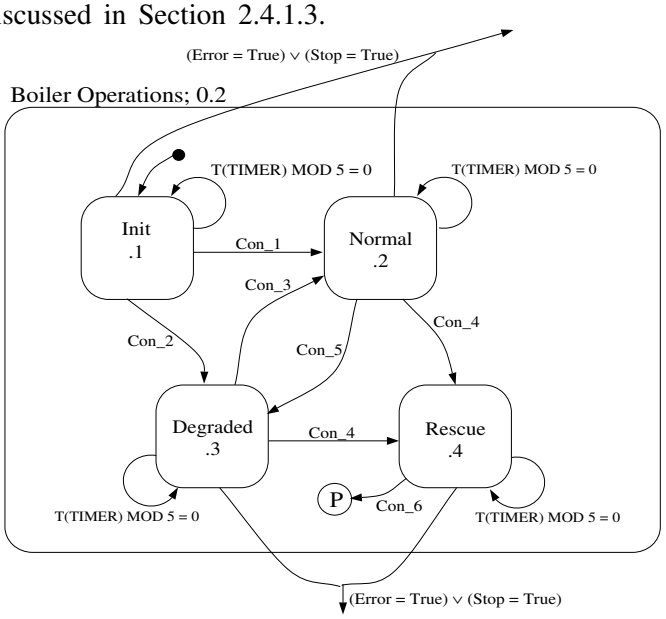


Figure 6: Common State

Running Example: Steam-boiler Controller The "Boiler Operations" state and the "Error Checking" state in Figure 5 can be further divided into more sub-states. Figure 7 shows the zoom-in of the "Boiler Operations" state. The "Boiler Operations" state contains four sub-states which represent four different modes of operation. The "Degraded" sub-state is the common state of the "Init" sub-state and the "Normal" sub-state, and the "Rescue" sub-state is the common state of the "Normal" sub-state and the "Degraded" sub-state. Note that the transitions lead to the common state are mutually exclusive (i.e., Con_2 and Con_5 cannot be true at the same time because S_Sta cannot be "_" and "Fail" simultaneously). Since the controller samples the data every five seconds, there is a self-looping transition with a transition condition "T(TIMER) MOD 5 = 0" for every sub-state in the "Boiler Operations" state. The T(arg) is a function which returns the current time of arg to the system. The circled P symbol is called a parent bubble which will be discussed in Section 2.4.1.3.



Con_1 ≡ (Phy_Unit_Ready = True) ∧ (∀i (0 ≤ i ≤ 3) P_Sta[i] = "_") ∧
(∀j (0 ≤ j ≤ 3) P_Con_Sta[j] = "_") ∧ (L_Sta = "_") ∧ (S_Sta = "_")

Con_2 ≡ (Phy_Unit_Ready = True) ∧ (L_Sta = "_") ∧ (S_Sta = "_") ∧
[(∃i (0 ≤ i ≤ 3) P_Sta[i] = Fail) ∨ (∃j (0 ≤ j ≤ 3) P_Con_Sta(j) = Fail)]

Con_3 ≡ ((∀i (0 ≤ i ≤ 3) P_Sta[i] ≠ Fail) ∧ (∀j (0 ≤ j ≤ 3) P_Con_Sta[j] ≠ Fail) ∧
(L_Sta ≠ Fail) ∧ (S_Sta ≠ Fail)

Con_4 ≡ (L_Det = Fail) ∧ (S_Det ≠ Fail)

Con_5 ≡ (L_Sta ≠ Fail) ∧ [(S_Sta = Fail) ∨ (∃i (0 ≤ i ≤ 3) P_Sta[i] = Fail) ∨
(∃j (0 ≤ j ≤ 3) P_Con_Sta[j] = Fail)]

Con_6 ≡ (L_Sta = Rep)

Figure 7: "Boiler Operation" State

In order to model the system more clearly, the "Error Checking" state is divided into two parallel actions which check the error related to the pump and pump controller units, and that related to the measuring units (Figure 8).

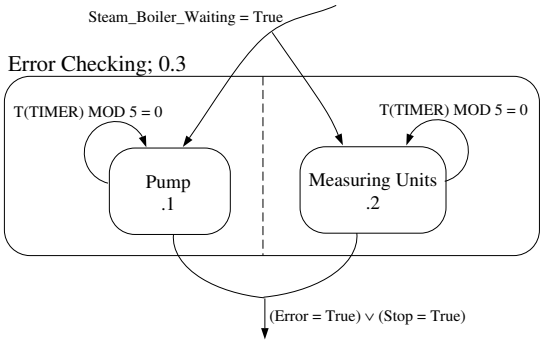


Figure 8: "Error Checking" state

2.4.1.2 Transition A transition is used to connect two states. Except for the entry transition, there is a transition condition associated with each of the transitions. A transition can be a normal transition, an entry transition, a non-entry transition, a selective transition, or a broadcast transition.

Normal Transition A normal transition is a basic transition which connects two state. If it points to the boundary of a super-state, it is always associated with the entry transition inside the super-state.

Entry Transition An entry transition is an open-ended transition which connects to an entry state. Other than the entry transition which points to the super-state in the first level (Level 0) of the ESD, there will not be any transition condition along any other entry transitions. This is because an entry state is activated by the transition terminating at the boundary of the super-state of this entry state. The entry transition which points to the super state in the first level of the ESD needs a transition condition because this entry transition is the very first transition in the system. For example, in Figure 9, when the super-state Y is activated by the transition condition δ, the sub-state A will be executed because it is connected with an entry transition.

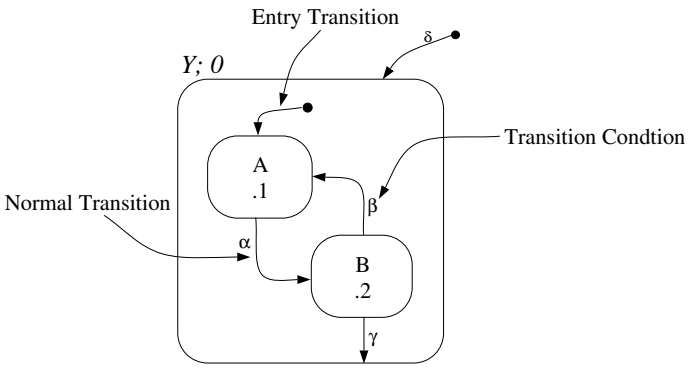


Figure 9: Entry Transition

Non-Entry Transition When a transition is not connected to the entry state of a super-state, it is called a non-entry transition. In the diagram, this transition terminates at a bar inside a super-state (Figure 10a). When the sub-state A is zoomed in, the state which is connected by the non-entry transition must be specified. For example, in Figure 10b, the transition β is a non-entry transition which is connected to substate D of the state A.

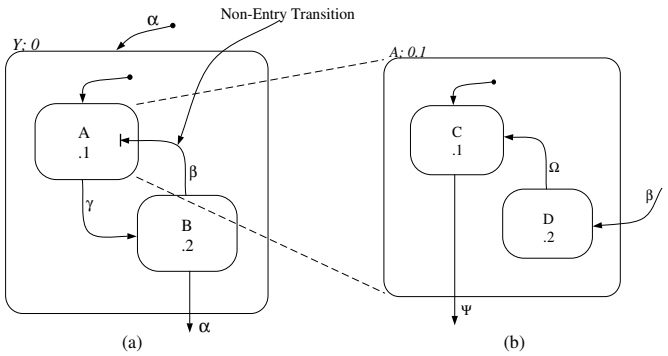


Figure 10: Non-Entry Transition

Running Example: Steam-boiler Controller In Figures 7 and 8, because of the sampling cycle, each sub-state has a self-looping transition with the transition condition "T(TIMER) MOD 5 = 0". The "Steam-boiler" concurrent state (Figure 5), however, does not show any information about the sampling cycle. In order to provide a complete picture of the system flow, a self-looping non-entry transition is added to the "Boiler Operations" sub-state and the "Error Checking" sub-state in the "Steam-boiler" concurrent state. Figure 11 is the revision of the "Steam-boiler" concurrent state.

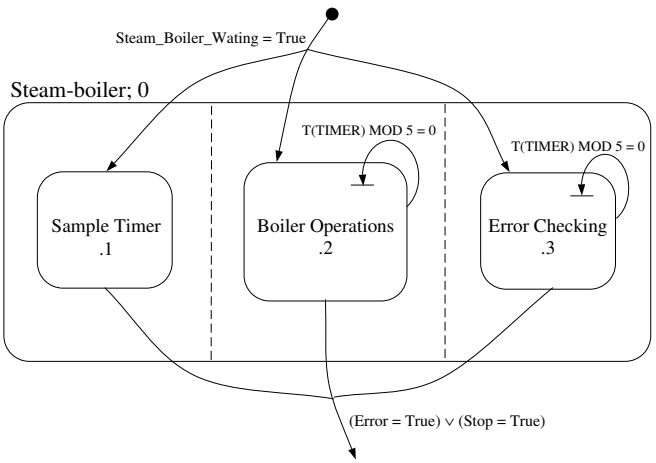


Figure 11: Revision of the "Steam-boiler" concurrent state

Broadcast Transition A broadcast transition either terminates at the boundary of a concurrent state, or originates from the boundary of a super-state or a concurrent state. A broadcast transition implies that the transition will be used by more than one state. Use of broadcast transitions enhances the clarity of presentation in the ESD. The notation of a broadcast transition in the ESD is a double-headed arrow.

When a broadcast transition terminates at the boundary of a concurrent state, it indicates that the transition is connected to all entry states in the concurrent state (Figure 12). The broadcast transition concept can be applied in any situation, as long as the transition is connected to all the entry states. A broadcast transition never terminates at a super-state because only one sub-state in the super-state can be executed at any time.

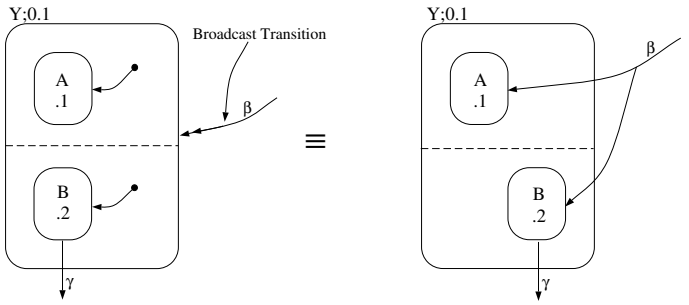


Figure 12: Broadcast transition points to a concurrent state

When a broadcast transition originates from the boundary of a super-state or a concurrent state, it implies that the transition actually originates from all the sub-states within the super-state or the concurrent state (Figures 13 and 14). A broadcast transition can be used if and only if all the transitions from the sub-states have the same transition condition (for example, α in Figure 13 and 14).

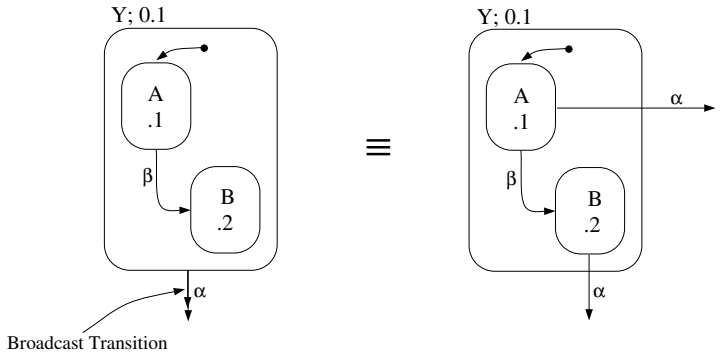


Figure 13: Broadcast transition originates from a super-State

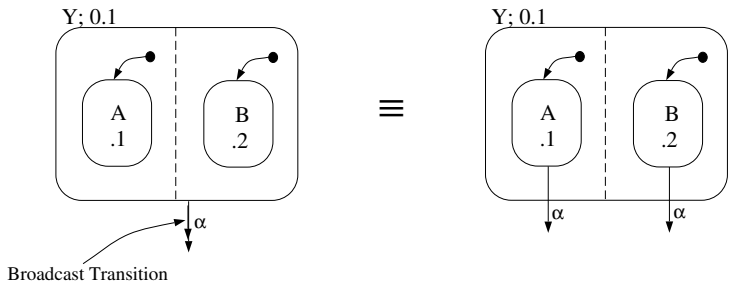


Figure 14: Broadcast transition originates from a concurrent state

Running Example: Steam-boiler Controller By using broadcast transitions and entry transitions, one can reduce the number of transitions used in Figures 7, 8, and 11. For example, in the "Boiler Operation" super-state (Figure 7), the transitions with the same transition condition "(Error = True) ∨ (Stop = True)" are connected to each of the sub-state. These transitions can be eliminated by using a broadcast transition which originates from the border of the "Boiler Operations" super-state. Figures 15, 16, and 17 are the revisions of the Figures 11, 7, and 8 respectively.

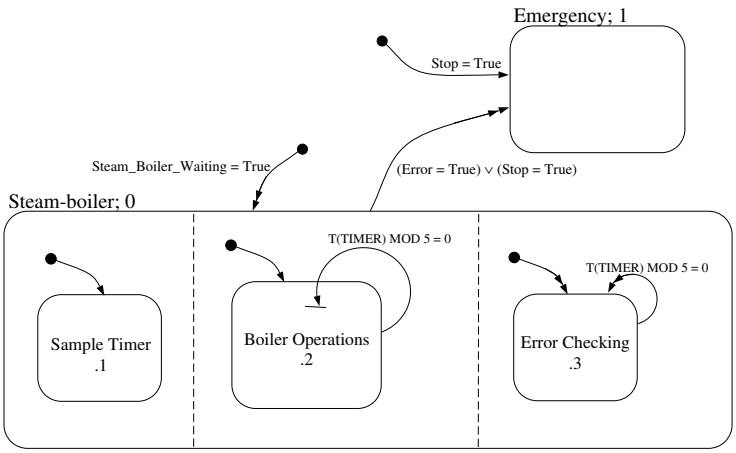


Figure 15: Revision of Level 0 diagram of the steam-boiler controller

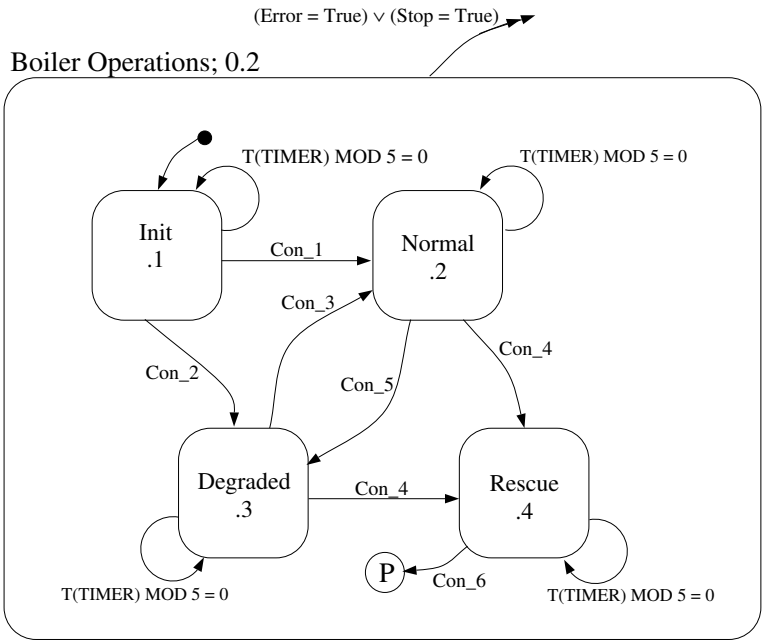


Figure 16: Revision of "Boiler Operations" state

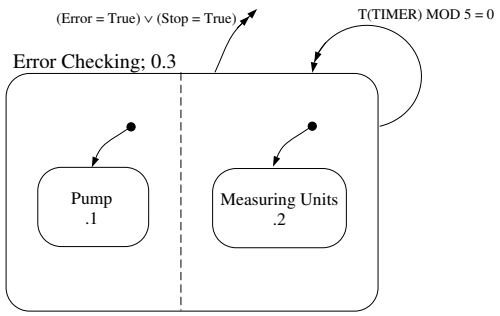


Figure 17: Revision of "Error Checking" state

Selective Transition In the ESD, it is possible to have more than one transition originating from a state. Such transitions are called selective transitions because, in the event that more than one transition becomes true simultaneously, only one transition can be selected. In order to identify which transition is selected, a priority number is added to each transition (Figure 18). If no priority numbers are given, the left-most transition is assumed to have the highest priority, and the right-most one will have the lowest priority.

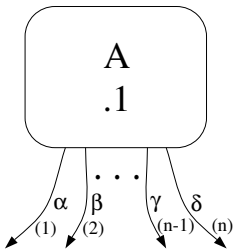


Figure 18: Selective Transition with Priority Number

Running Example: Steam-boiler Controller Most of the sub-states in Figures 15,16, and 17 have at least two exiting transitions (i.e., the transitions originate from a sub-state). A priority number should be given to every selective transition, even though some of the transitions are mutually exclusive, or it is unlikely that the transition will become true simultaneously is very small. By doing so, readers will have a clear and unambiguous picture of the system state transition. Figures 19, 20, and 21 are the final versions of ESD diagrams of the steam-boiler controller.

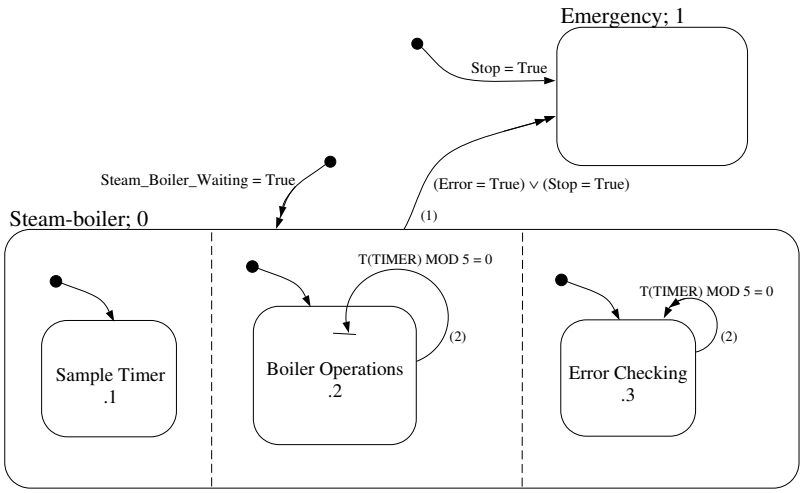


Figure 19: Final version of Level 0 diagram of the steam-boiler controller

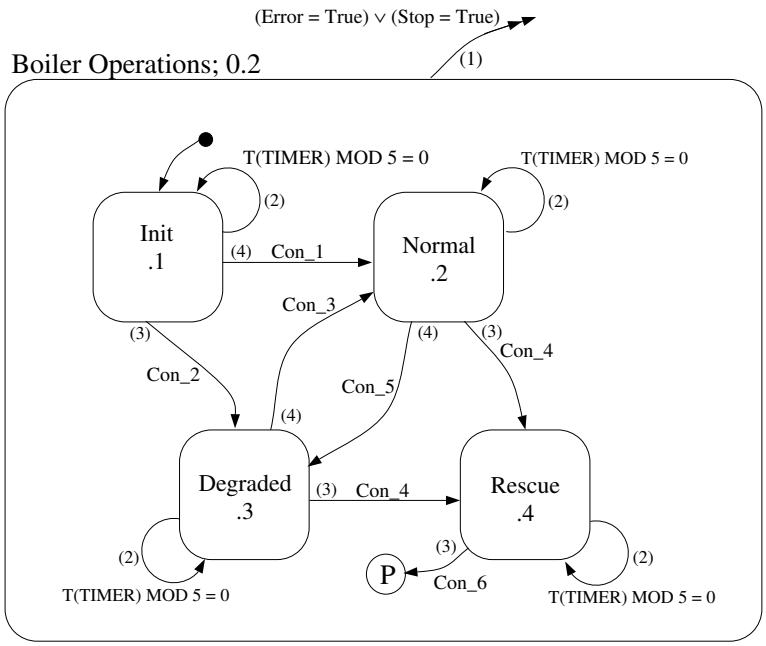


Figure 20: Final version of "Steam Operations" sub-state

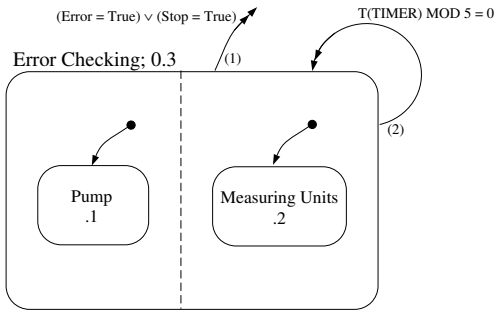


Figure 21: Final version of "Error Checking" sub-state

2.4.1.3 Parent Bubble In a conventional state diagram, it is very difficult to specify a common state which is connected back to its previous states by the transitions which have the same transition condition (Figure 22a). This backward state transition is represented by a transition pointing to a circled P in the ESD (Figure 22b). This circled P is called a parent bubble. It possesses the advantage of reducing the number of transitions needed in the diagram. The parent bubble can be used in an ESD diagram if and only if it is connected to a common state and all the exiting transition conditions from the common state are the same.

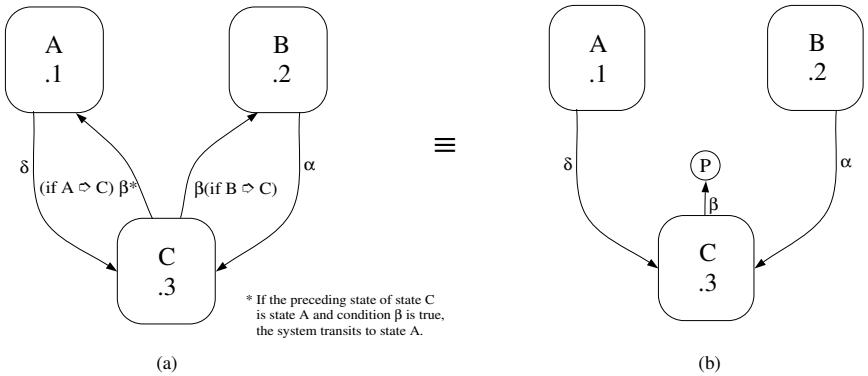


Figure 22: Parent Bubble

Running Example: Steam-boiler Controller Once the physical unit is fixed in the "Rescue" sub-state in Figure 20, the system will go back to either "Normal" sub-state or the "Degraded" sub-state, depending on which one is the preceding sub-state of the "Rescue" sub-state. This situation is modelled by a parent bubble in Figure 20.

2.4.2 State Specification (S-Spec) The state specification (S-Spec) is the lowest level specification in the ESD. In other words, when a sub-state cannot be decomposed any further (it is called a primitive state), it is described by a S-Spec. A S-Spec consists of a State Action Table (SAT) and a Control Action Table (CAT). Except for primitive entry states in a concurrent state, each primitive state must have its own pair of SAT and CAT. All primitive entry states in a concurrent state can be specified by one SAT and a set of CATs because all the primitive entry states have the same state condition.

When a primitive state is activated by a system, its SAT will always be executed. Then, depending on the action specified in the SAT, the system will either assign a value to a controlled value, or execute the CAT of the current primitive state, or both.

2.4.2.1 SAT The SAT is like a simple finite state machine in a primitive state. It specifies the immediate relationship between the internal states and the monitored variables. This relationship only occurs at the moment of the table being activated. In other words, the SAT is only valid at the moment that a primitive state is activated because the value of the monitored variables is changing constantly. The SAT of a primitive state must be executed if this primitive state is currently activated.

The row heading contains the internal states, which must be primitive and self-explanatory. The column headings are the state conditions, which are formed by the monitored variables. The cells specify the action and the next state (Figure 1). The format of the cells is ACTIONS/NEXT_STATE. In the ACTIONS field, multiple actions can be specified; however, only one next state can be specified in the NEXT_STATE field.

A SAT is complete if and only if all cells are filled and the state conditions have an opposite relationship. For example, if one of the state condition is "Switich_1 = True", another condition must be "Switch_1 = False".

	State Conditions	
Internal States		
	Actions/Next State	Actions/Next State
	Actions/Next State	Actions/Next State

Table 1: Format of SAT

A cell in the table can be executed if and only if its corresponding state condition is true and the corresponding internal state is the current system internal state. Once the cell is executed, the ACTIONS will be carried out first and then the internal state will be changed to the NEXT_STATE. The ACTIONS could be used to call the corresponding CAT, to assign a retentive value to a controlled variable (is called assignment action), or both. When there are multiply assignment actions, they will be executed simultaneously because all actions have the same state condition. Since the controlled variables in the assignment actions can be used as the conditions in the corresponding CAT, the assignment actions and the CAT must be executed sequentially if they are placed together in the ACTIONS field. The state specified in the NEXT_STATE field must be one of the internal states in the row heading of the current SAT table.

The state condition must specify the expected value of all monitored variables at the moment the state is entered. If there are a large number of monitored variables, the state condition will become lengthy and complicated. Two functions are defined in order to make the state condition simpler and clearer. The first function is used to indicate that a new value has been assigned to the monitored variable at the immediate preceding state but has not been changed during the state transition. This function is defined as follow: $UC(m_0, m_1..., m_n) \equiv (m_0 = x_0) \wedge (m_1 = x_1) \wedge ... \wedge (m_n = x_n)$, where $m_0, m_1..., m_n$ are the monitored variables, and $x_0, x_1, ..., x_n$ are the values that have been assigned to the monitored variables at the immediate preceding state. The function is useful when the current state is a common state, and the preceding states assign different values to the monitored variables. This function also helps one to discover any error during the state transition. For example, when the state condition, $UC(m_0, m_1)$, is

being checked in the current, if the value m_0 is not equal to the its assigned value at the immediate predecessor state, one will know that there is an error during the state transition.

The second function is used to specify that all monitored variables stated in the function are untouched by the immediate preceding state. In other words, the preceding state has not assigned any new values to the monitored variables. This function is defined as follow: $UT(m_0, m_1..., m_n) = (m_0 = y_0) \wedge (m_1 = y_1) \wedge ... \wedge (m_n = y_n)$, where $m_0, m_1..., m_n$ are the monitored variables, and $y_0, y_1, ..., y_n$ are the most recent values assigned to the monitored variables. One can use this function in the current state condition when the value of a monitored variable has not been changed at the immediate preceding state.

Running Example: Steam-boiler Controller Table 2 is the SAT of the "Normal" sub-state in the steam-boiler controller. Normal_Con is defined outside the table because there is not enough space to contain the whole definition within the table. The internal states of the "Normal" sub-state are "idle", "correct", and "error". It is assumed that the system is in the "idle" internal state when the "Normal" sub-state is activated at the very first time. By using the UC function, if the value of the variables specified in the function are changed during the state transition, it will cause an error in the system.

SAT: Normal; 0.2.2

$Let\ Normal_Con \equiv UT(Steam_Boiler_Waiting, Stop, Error) \wedge UC(P_Det, P_Con_Det, L_Det, S_Det, P) \wedge (Phy_Unit_Ready = True) \wedge (Valve = Close)$

	Normal_Con = True	Normal_Con = False
idle	/correct	/error
correct	CAT/idle	/error
error	/correct	"Error := True"/idle

Table 2: SAT for the "Normal" sub-state

2.4.2.2 CAT This table is used to specify the relationship between controlled variables and monitored variables. The controlled variables and the monitored variables (conditions) are placed in the row headings and the column headings of the table respectively. The cells are used to specify the resulting values of the controlled variables (Figure 3). This table can be executed if and only if the action "CAT" in the SAT is activated.

		Conditions	
Controled Variables			
		Value	Value
		Value	Value

Table 3: Format of CAT

The value specified in the cell is assigned to the corresponding controlled variable if and only if the condition is true and the table has been activated by the SAT. The value assigned to the controlled variable is retained until a different value is reassigned by a different condition. Note that only one value can be assigned to a controlled variable at any time; however, more than one controlled variable can be assigned to a new value simultaneously. A CAT is complete if and only if all cells are filled and the conditions have the opposite relationship.

Running Example: Steam-boiler Controller Depending on the current water level, the controller will either turn on or turn off all the pumps. If the water level is not between the limited value, M_1 and M_2, the system will assign a true value to the controlled variable "Error".

CAT: Normal; 0.2.2

	$M_1 \leq Level \leq N_1 + 5W$	$N_2 \leq Level \leq M_2$	$(Level > M_2) \vee (Level < M_1)$
P :=	On	Off	no_op
Error :=	no_op[1]	no_op	True

Table 4: CAT for the "Normal" sub-state

2.4.2.3 S-Spec for a Concurrent State All the primitive entry states can be specified by one special S-Spec, owing to the fact that these states must have the same state condition and internal states. In other words, there is only one SAT for these primitive entry states. One can either use a special CAT or a set of normal CATs to specify the assignments in each entry state. This special CAT has one extra column, allocated at the right-hand side of the normal CAT, to distinguish the assignments for each entry state (Table 5). When an entry state is not activated, the assignments located in the same row as the entry state will not be executed. However, the assignments in other entry states will be executed as long as they are active. If one wants to use a set of normal CATs to specify the assignments, each entry state will have its own normal CAT.

	Conditions		
Controled Variables	Value	Value	Entry States or Concurrent State
	Value	Value	

Table 5: Format of a Special CAT

[1] no_op indicates that the system does not change the value in the corresponding controlled variable in this row.

Running Example: Steam-boiler Controller The "Sample Timer" sub-state, the "Init" sub-state, the "Pump" sub-state, and the "Measuring Units" sub-state are specified by one SAT and four CATs because these sub-states are executed simultaneously. The special CAT is not used here because the conditions in the CATs of the sub-states are not related to each other.

The complete S-Spec of the steam-boiler controller is in Appendix B.

2.5 Conclusions

GrafTab is an innovative method for writing a requirements specification for a control system. The features in *GrafTab* are summarized as follows:

— **Readable**: Graphical (in the ESD) and tabular (in the S-Spec) description techniques are used in writing the system requirements document in *GrafTab*; thus the specification is highly readable.

— **Precise**: In specifying a primitive state with a S-Spec, one is forced to specify the internal state condition and the actions completely. This results in a very precise specification.

— **Complete**: *GrafTab* offers a complete requirements specification for a control system. All necessary information is specified in the *GrafTab* documents. Firstly, the overall function of the system is described in the system abstract. Secondly, the environmental variables are identified in the monitored and controlled variables specification and in the external interface document. Finally, the flow of the system states and the relationships among the environmental variables are specified carefully in the system requirements document.

Since each cell in S-Spec tables must be filled, a S-Spec not only provides a complete specification of the actions in a primitive state, but also no ambiguity would be found in the tables.

— **Learnable**: The syntax of *GrafTab* is comprehensible. Conventional and simple notations are used. Only limited training is needed for using *GrafTab*.

— **Powerful**: *GrafTab* is a powerful tool for specifying system states. By using the concurrent state notations in the ESD, one can specify parallel actions in the system. Broadcast transitions can reduce the number of transitions needed in the ESD diagrams substantially. A parent bubble records the history of state transitions.

2.5.1 Cost of using *GrafTab*

Firstly, although it will take longer time to write a requirements specification in *GrafTab* than in English, the writing time will be offset by the decreased maintenance time and programming time of the system. Secondly, although the notations in the ESD help reduce the size of the state diagram, the state diagram will be really big if a control system is very complicated. Finally, since each primitive state in an ESD specification has to be specified by a S-Spec, the specification tends to be lengthy.

2.5.2 Steam-boiler Controller

The requirements specification of the steam-boiler controller in this paper is precise, complete and easy to read. *GrafTab* uses graphs to describe the control flow of the steam-boiler controller, and tables to specify the actions in each primitive state. Once the readers understand

the syntax of *GrafTab*, they will capture the control flow of the system in very short time. From Level 0 ESD diagram, readers know that the system contains two major states, namely “Steam-boiler” and “Emergency”. The details of the system are then revealed by zooming into these two states. When the readers reach the primitive states, the actions and the state condition of each primitive state are specified by the tables in a S-Spec. Completeness of the requirements specification of the steam-boiler controller is increased because filling of every cell in the tables and specification of the state condition are compulsory.

Bibliography

[1] Jean-Raymond Abrial. Steam-boiler control specification problem. In *Dagstuhl Meeting: Method for Semantics and Specification*, August 1994.

[2] David Harel. Statecharts: A Visual Formalism for Complex System. *Sc. Comp. Prog.*, 8:231–274, 1987.

Appendix A Monitored and Controlled Variables Specification of the Steam-boiler Controller

Controlled Variables

Program_Ready

— signal for indicating that the system is ready

— possible values: {True, "_"}

— initial value: "_"

Valve

— valve for evacuation of water from the steam-boiler

— possible values: {Open, Close}

— initial value: Close

P[0..3]

— each element of the array represents a pump which provides the steam-boiler with water

— possible values: {On, Off}

— initial value: Off

P_Det[0..3]

— each element of the array represents a signal for indicating that the program has detected an error in the corresponding pump, or the program has received a repaired acknowledgment from the corresponding pump

— the index of this array corresponds to the index in the variable P (i.e., P_Det[x] is a message for the P[x] where x = 0,1,2,3)

— possible values: {Fail, Repaired, "_"}

— initial value: "_"

P_Con_Det[0..3]

— each element of the array represents a signal for indicating that the program has detected an error in the corresponding pump controller, or the program has received a repaired acknowledgment from the corresponding pump controller

— the index of this array corresponds to the index in the variable P (i.e., P_Con_Det[x] is a message for the controller of the P[x] where x = 0,1,2,3)

— possible values: {Fail, Repaired, "_"}

— initial value: "_"

L_Det

— signal for indicating that the program has detected an error in the water level measuring unit, or the program has received a repaired acknowledgment from the water level measuring unit
— possible values: {Fail, Repaired, "_"}
— initial value: "_"

S_Det

— signal for indicating that the program has detected an error in the steam outcome measuring unit, or the program has received a repaired acknowledgment from the steam outcome measuring unit
— possible values: {Fail, Repaired, "_"}
— initial value: "_"

Error

— internal signal for indicating that the program has detected an error which will cause an emergency stop
— possible values: {True, False}
— initial value: False

TIMER

— internal timer for sampling the data
— possible values: {On, Off}
— initial value: Off

Emergency_Light

— emergency light
— possible values: {On, Off}
— initial value: Off

Monitored Variables

Stop

— emergency stop signal
— possible values: {True, False}

Steam_Boiler_Waiting

— message from the steam-boiler
— possible values: {True, "_"}

Phy_Unit_Ready

— signal from the physical units
— possible values: {True, "_"}

P_S[0..3]

— each element of the array represents a signal for indicating that the state of the corresponding pump
— the index of this array corresponds to the index in the controlled variable P (i.e., P_S[x] is a message from the P[x] where x = 0,1,2,3)
— possible values: {Open, Close}

P_Con_S[0..3]

— each element of the array represents a signal for indicating that the state of the corresponding pump controller
— the index of this array corresponds to the index in the controlled variable P (i.e., P_Con_S[x] is a message from the P[x] where x = 0,1,2,3)
— possible values: {Flow, No_Flow}

Level

— quantity of water in the steam-boiler
— possible values: Real (liter)

Steam

— quantity of steam exiting the steam-boiler
— possible values: Real (liter/sec)

P_Sta[0..3]

— each element of the array represents a signal for indicating that the corresponding pump has been repaired, or the corresponding pump has received a failure detection signal from the program
— the index of this array corresponds to the index in the controlled variable P (i.e., P_Sta[x] is a message for the P[x] where x = 0,1,2,3)
— possible values: {Rep, Fail, "_"}

P_Con_Sta[0..3]

— each element of the array represents a signal for indicating that the corresponding pump controller has been repaired, or the corresponding pump controller has received a failure detection signal from the program
— the index of this array corresponds to the index in the controlled variable P (i.e., P_Sta[x] is a message for the P[x] where x = 0,1,2,3)
— possible values: {Rep, Fail, "_"}

Level_Sta

— signal for indicating that the water level measuring unit has been repaired, or the water level measuring unit has received a failure detection signal from the program
— possible values: {Rep, Fail, "_"}

Steam_Sta

— signal for indicating that the steam outcome measuring unit has been repaired, or the steam outcome measuring unit has received a failure detection signal from the program

— possible values: {Rep, Fail, "_"}

Appendix B S-Spec of the Steam-boiler Controller

Table 6 shows the physical variables of the system used in the S-Spec. These variables are the constrain of the physical units. This table is not a part of the S-Spec.

Variable Name	*Unit*	*Comment*
		Quantity of water in the steam-boiler
C	liter	Maximal capacity
M_1	liter	Minimal limit
M_2	liter	Maximal limit
N_1	liter	Minimal normal
N_2	liter	Maximal normal
		Outcome of steam at the exit of the steam-boiler
W	liter/sec	Maximal quantity
U1	liter/sec/sec	Maximum gradient of increase
U2	liter/sec/sec	Mximum gradient of decrease
		Current measures
p	liter/sec	Throughput of the pump

Table 6: Physical variables of the steam-boiler controller

S-Spec

SAT: Steam-boiler; 0

$Let\ Steam_boiler_Con \equiv (Steam_Boiler_Waiting = True) \wedge (Stop = False) \wedge (Error = False) \wedge UC(Phy_Unit_Ready, P_Det, P_Con_Det, L_Det, S_Det, Valve)$

	Steam_bolier_Con = True	Steam_boiler_Con = False
idle	/correct	/error
correct	CAT/idle	/error
error	/correct	"Error := True"/idle

CAT: Sample Timer; 0.1

	$(Error = False) \wedge (Stop = False)$	$(Error = True) \vee (Stop = True)$
TIMER :=	On	Off

CAT: Init; 0.2.1

	$(Steam \neq 0) \vee (Level < 0) \vee (Level > C) \vee$ $(Level > M_2) \vee (Level < M_1) \vee$ $[(\exists_{i(0 \leq i \leq 3)} P[i] = On) \wedge$ $\left(('Level \geqslant Level) \vee \left('Level + 5 \sum_{i=0}^{3} p_i < Level \right) \right)] \vee$ $[(\forall_{i(0 \leq i \leq 3)} P[i] = Off) \wedge$ $(('Level \leq Level) \vee ('Level - 5W > Level))]$[2]	$N_2 < Level < M_2$	$M_1 < Level < N_1 + 5W$[3]	$N_1 + 5W \leq Level \leq N_2$
Error :=	True	no_op	no_op	no_op
Valve :=	no_op[4]	Open	Close	Close
Pump :=	no_op	Off	On	Off
Program_Ready :=	no_op	no_op	no_op	True

[2] $'Level$ indicates the water level value in the previous sampling cycle. This applies to all other variables which have the quotation signs before them.
The equations in the square brackets refer to the dynamics error of the system. The first equation indicates that if some of the pumps are on, the current water level must be greater than its previous sampling value, or the current water level must be less than or equal to the previous sampling water level plus the water throughput of the pumps in 5 seconds. The second equation indicates that if all pumps are broken, the current water level must be less than its previous sampling value, or the current water level must be greater than or equal to the previous water level sampling value minus the maximun steam output in 5 seconds.

[3] The maximun steam output in 5 seconds is taken into account here.

[4] no_op indicates that the system does not change the value in the corresponding controlled variable in this row.

CAT: Pump; 0.3.1

	$\exists_{i(0\leq i\leq 3)}[('P[i] = On) \wedge$[5]		$\exists_{i(0\leq i\leq 3)}[('P[i] = Off) \wedge$		$\exists_{i(0\leq i\leq 3)}(P_Sta[i] = Rep)$	$\exists_{i(0\leq i\leq 3)}(P_Con_Sta[i] = Rep)$
	$(P_S[i] = Off) \wedge ('P_Det[i] \neq Fail)]$	$(P_Con_S[i] = No_Flow) \wedge ('P_Con_Det[i] \neq Fail)]$	$(P_S[i] = On) \wedge ('P_Det[i] \neq Fail)]$	$(P_Con_S[i] = Flow) \wedge ('P_Con_Det[i] \neq Fail)]$		
P_Det[i] :=	Fail	no_op	Fail	no_op	Repaired	no_op
P_Con_Det[i] :=	no_op	Fail	no_op	Fail	no_op	Repaired

CAT: Measuring Units; 0.3.2

	$('LDet \neq Fail) \wedge \{(Level < 0) \vee (Level > C) \vee [\exists_{i(0\leq i\leq 3)}(P[i] = On) \wedge (('Level > Level) \vee \left('Level + 5\sum_{i=0}^{3} p_i < Level\right))] \vee [\exists_{i(0\leq i\leq 3)}(P[i] = Off) \wedge (('Level < Level) \vee ('Level - 5W > Level))]\}$	$('S_Det \neq Fail) \wedge [(Steam < 0) \vee (Steam > W) \vee \left(\dot{Steam} > u_1\right) \vee \left(\dot{Steam} < -u_2\right)]$	$Level_Sta = Rep$	$Steam_Sta = Rep$
L_Det :=	Fail	no_op	Repaired	no_op
S_Det :=	no_op	Fail	no_op	Repaired

[5] This is a partial condition. The complete condition contains this partial condition and the partial condition in one of the heading cell right below this cell. For example, the complete condition here is either $\exists_{i(0\leq i\leq 3)}[('P[i] = On) \wedge (P_S[i] = Off) \wedge ('P_Det[i] \neq Fail)]$ or $\exists_{i(0\leq i\leq 3)}[('P[i] = On) \wedge (P_Con_S[i] = No_Flow) \wedge ('P_Con_Det[i] \neq Fail)]$. This rule applies to all other heading cells which have the same format as this one.

SAT: Normal; 0.2.2

$Let\ Normal_Con \equiv UT(Steam_Boiler_Waiting, Stop, Error) \wedge UC(P_Det, P_Con_Det, L_Det, S_Det, P) \wedge$
$(Phy_Unit_Ready = True) \wedge (Valve = Close)$

	Normal_Con = True	Normal_Con = False
idle	/correct	/error
correct	"Program_Ready := "_"", CAT/idle	/error
error	/correct	"Error := True"/idle

CAT: Normal; 0.2.2

	$M_1 < Level < N_1 + 5W$	$N_2 < Level < M_2$	$(Level > M_2) \vee (Level < M_1)$
P :=	On	Off	no_op
Error :=	no_op	no_op	True

SAT: Degraded; 0.2.3

$Let\ Degraded_Con \equiv UT(Steam_Boiler_Waiting, Stop, Error, Program_Ready, Phy_Unit_Ready, Valve) \wedge$
$UC(P, P_Det, P_Con_Det, S_Det) \wedge$
$(L_Det \neq Fail) \wedge (L_Sta \neq Fail)$

	Degraded_Con = True	Degraded_Con = False
idle	/correct	/error
correct	"Program_Ready := "_"", CAT/idle	/error
error	/correct	"Error := True"/idle

CAT: Degraded; 0.2.3

	$M_1 < Level < N_1 + 5W$	$N_2 < Level < M_2$	$(Level > M_2) \vee (Level < M_1)$
P :=	On	Off	no_op
Error :=	no_op	no_op	True

SAT: Rescue; 0.2.4

$Let\ Rescue_Con \equiv UT(Steam_Boiler_Waiting, Stop, Error, Program_Ready, Phy_Unit_Ready, Valve) \wedge$

$UC(P, P_Det, P_Con_Det, L_Det) \wedge$

$(S_Sta \neq Fail) \wedge (S_Det \neq Fail)$

	Rescue_Con = True	Rescue_Con = False
idle	/correct	/error
correct	CAT/idle	/error
error	/correct	"Error := True"/idle

CAT: Rescue; 0.2.4

	$M_1 <' Level + 5\sum_{i=0}^{3} p_i - 5Steam < N_1 + 5W$	$N_2 <' Level + 5\sum_{i=0}^{3} p_i - 5Steam < M_2$	$(Steam < 0) \vee (Steam > W) \vee$ $\left(\dot{Steam} > u_1\right) \vee \left(\dot{Steam} < -u_2\right) \vee$ $(\exists_{i(0 \leq i \leq 3)}(('P[i] = On) \wedge (P_Con[i] = No_Flow))) \vee$ $(\exists_{i(0 \leq i \leq 3)}(('P[i] = Off) \wedge (P_Con[i] = Flow)))$
P :=	On	Off	no_op
Error :=	no_op	no_op	True

SAT: Emergency; 1

	Any Condition
idle	/error
correct	/error
error	CAT/correct

CAT: Emergency; 1

	Any Condition
Program_Ready:=	"_"
Valve :=	Close
P :=	Off
P_Det :=	no_op
P_Con_Det :=	no_op
L_Det :=	no_op
S_Det :=	no_op
TIMER :=	Off
Emergency_Light :=	On