EXCEPTIONS AND CONTINGENCIES HANDLING IN A SCADA SYSTEM

by

Rekha Arora

Submitted in partial fulfillment of the
requirements for the degree of
Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
January 2011

DALHOUSIE UNIVERSITY

FACULTY OF COMPUTER SCIENCE

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled "EXCEPTIONS AND CONTINGENCIES HANDLING IN A SCADA SYSTEM" by Rekha Arora in partial fulfillment of the requirements for the degree of Master of Computer Science.

Dated: January 14, 2011

Supervisors: _____

_____

Reader: _____

DALHOUSIE UNIVERSITY

DATE: January 14, 2011

AUTHOR: Rekha Arora

TITLE: EXCEPTIONS AND CONTINGENCIES HANDLING IN A SCADA SYSTEM

DEPARTMENT OR SCHOOL: Faculty of Computer Science

DEGREE: M.C.Sc. CONVOCATION: October YEAR: 2011

Permission is herewith granted to Dalhousie University to circulate and to have copied for non-commercial purposes, at its discretion, the above title upon the request of individuals or institutions.

_____

Signature of Author

Table of Contents

List of Tables

List of Figures

Abstract

The use of rollback is a fundamental flaw in some existing distributed control systems because the advance in time and in external world situations means that what had been a correct state in the past may no longer be a correct state in real time and distributed systems. In such systems rollback is not restoring to a state that is consistent with the current external environment. Forward error recovery provides a potential solution to such a situation to handle exception rather than backward recovery. A contingency is an unusual but anticipated situation for which the normal flow of instructions would not produce the appropriate results that should be expected. We will discuss how to handle contingencies and exceptions in a SCADA (Supervisory Control and Data Acquisition) system using resumption and termination models of exception handling.

Contingencies are a consequence of the external world, independent of the program code. Rather than thinking of contingencies as a possible program error, contingencies instead consider as a rare but understood external situation which changes what the software would be expected to do from what is needed in the normal case. A SCADA system is a computer system sensing and controlling some large physical system and its environment. We attempted to handle contingencies in a specific SCADA system at Halifax water plant to modify the SFC (Sequential Flowchart) algorithm of Halifax Coarse Screen System.

List of Abbreviations Used

A to D	Analog to Digital
ACID	Atomicity, Consistency, Isolation, Durability
ACK_MODE	Acknowledgment mode
AO/AI	Analog Output/Analog Input
ASCII	American Standard Code for Information Interchange
BF I/O	Boolean Flag Input/Output
CA/CAA	Coordinated Atomic Actions
CALC	Calculation/Logic Function Block
CL	Close Command
CND	Condition Function Block
DC I/O	Device Control Input/Output
DCS	Distributed Control System
DI	Discrete Input
DLIT	Differential level of Input Terminal
DNP	Distributed Network Protocol
DO/DI	Discrete Output/Discrete Input
DTE	Data Terminal Equipment
ESD	Emergency shutdown
GUI	Graphical User Interface
HMI	Human Machine Interface
I/O	Input Output
I/P	Input
IBM	International Business Machines
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
LCD	Liquid Crystal Display

LOCO	Loco mode
LSH	Level Safety High
MAN	Manual mode
MTU	Master Terminal Unit
OLE	Object Linking and Embedding
OP	Output Command
O/P	Output
OPC	OLE Process Control
PLC	Programmable Logic Control
POS	Position of sensor
PV	Position Value
REM	Remote Manual
RET	Retrieve Timer
RTU	Remote Terminal Unit
SA	Sequential Arm
SCADA	Supervisory Control and Data Acquisition System
SFC	Sequential Flowchart
SIS	Software Intensive System
SOA	Service Oriented Architecture
TAR_MODE	Target Mode
VA	Valve Auto/Manual
VS	Safety Valve
WSDK	Web Services Development Toolkit

Acknowledgements

I sincerely appreciate the efforts of Dr. Morven Gentlemen for his continued support and steady guidance throughout this project. I am also thankful to Dr. Peter Bodrik, Dr. Michael McAllister for serving on my guiding committee and provide valuable suggestions to improve the quality of my work.

I extend special thanks to Alana Murray and other staff of Halifax Water for their help and use of the facilities at Halifax Wastewater Treatment Facility. A few visits to this facility with the cooperation of Halifax Water staff gave me in depth knowledge of the DCS(Distributed Control System) systems and their applications. Finally, I wish to thank my parents, my family in India and Canada for their continuous support and encouragement.

Chapter 1

Introduction

1.1 Exceptions and Contingencies

An exception is defined as "an event which occurs during the execution of a program that disrupts the normal flow of the program's instructions" [1]. It is an unanticipated event that occurs during the execution of a program and disrupts the normal flow of instruction. There is no connotation that "exception" implies "error". This definition highlights the arbitrariness of what the designer considers a "normal" or an "unexceptional" behavior and what is considered "exceptional".

A contingency is an unusual but anticipated situation for which the normal flow of instructions would not produce the appropriate results that should be expected, so the normal flow of instructions must be disrupted in order to instead execute the relevant instructions. According to Van Ellen et al., "A contingency is an unusual but anticipated situation for which the normal flow of instructions must be disrupted because the results (if any) that would be produced by the normal flow of instructions would not be appropriate to this situation" [2].

Van Ellen and Hasselbring contend that contingencies are expected and unusual situations, whereas exceptions are unexpected or unanticipated situations. Moreover, contingencies are not errors because they are not part of a specification violation. Thus, a situation that is neither an error nor an exception but is unusual, anticipated, disrupts the normal flow, and prevents the expected operations, is a contingency. The main distinction is that it arises when the external environment situation is changed. Examples include paper jamming in a printer, the same value (green or red) being displayed by a traffic light signal at an intersection, and an outgoing fax encountering a busy line.

Contingencies were introduced by Van Ellen and Hasselbring[2] as a completely different perspective on a possible exception. Rather than thinking of contingencies as a possible program error, Van Ellen and Hasselbring[2] instead consider contingencies

as a rare but understood external situation which changes what the software should be expected to do from what is needed in the normal case. This can be expressed in a richer specification and is not related to any particular code implementation. Indeed, detection of the contingency need not be by conventional exception mechanisms, although that may be the most efficient and highest performance implementation.

Goodenough[3] states that the two ways to handle exceptions are forward recovery and backward recovery. We will explain how the term contingency differentiates from exception. Goodenough[3] described that exceptions and exceptions handling are interleaving actions that do not simply deal with errors; rather, they are levels of abstractions. If a program is not producing the expected results, we can change the normal flow of control of the program based on another set of instructions defined in an exception handler block, and make the decision to transfer the control from normal flow and expected instructions based on the intermediate results, thereby saving the effort of partial computation using resumption. Goodenough[3] states that forward recovery is the best technique for handling exceptions and contingencies in real time and distributed systems. In many systems, what had been the correct state in the past may no longer be a valid state because external conditions have changed.

1.2 Problem Definition

The main problem is how to handle contingencies. The crux of the issue is that rollback is a fundamentally flawed idea in some existing systems because the advance in time and in external world situations means that what had been a correct state in the past may no longer be a valid state. In such systems rollback is not restoring to a state that is consistent with the current external environment. If servers are subject to client requests from outside the application being studied, restoring that application to a previous state generally does not restore it to a correct state because the servers are outside the control of the application itself and so their state cannot be restored. Contingencies, as introduced by Van Ellen and Hasselbring [2], can be expressed in a richer specification and are not related to any particular code implementation. Resumption means disregarding past history and moving forward to a state which is known to be safe and from which processing can continue. This safe state depends on current values but need not be any state previously experienced. Resumption does

provide a potential solution to such a situation, but there are few detailed examples of exception handling leading to effective forward error recovery or resumption. Most papers are general and generic, few deal with resumption, and fewer still give concrete realistic examples. There are certain conditions that can disrupt the normal flow of the system, but these conditions can be dealt with at run time. Such conditions are called contingencies not exceptions.

According to Van Ellen, " A contingency is a situation that is described within the specification of a module, and represents a module result where the task or function, which calling modules depend on, was not performed. Contingencies differ from normal situations in that normal situations do not represent work refusals and without additional specific measures do not necessarily run into errors" [2].

In this work we will identify contingencies and handle them by first detecting the state of variables pertinent to the contingency and then performing either a roll-back to a safe state or resume the valid state using resumption, as is appropriate.

1.3 Objective

The objective of this thesis is to demonstrate how the contingency concept used with resumption and termination can effectively address important risks in a SCADA system, which is a real time and embedded system. Some contingencies, such as those that can produce damage if left unattended, must be handled. They can be identified in advance at a specification level or they can be handled at implementation phase if they are not detected at the specification level [4, 5].

We will demonstrate how to handle contingencies in a SCADA (Supervisory Control and Data Acquisition) system. A computer system sensing and controlling some large physical system and its environment is known as SCADA System. We will discuss to handle contingencies in a specific SCADA at Halifax water plant to modify the SFC (Sequential Flowchart) algorithm of the "Halifax Coarse Screen" System.

We will deem our contingencies handling attempt as successful if it is able to convince the staff of the Halifax Water Plant that it is effective. We will detect the unusual situations that are effected by external environment changes and attempt to handle them using resumption and termination. We will record the amount of effort to apply our attempt in order to provide guidance as to what might be involved in

treating other contingencies.

1.4 Thesis Outline

This chapter discussed contingencies and exceptions and presented the objective of the thesis. In the next chapter, we provide some relevant background on the SCADA system in question, which is a real-time, embedded distributed system. Furthermore we review relevant literature on exceptions, termination models, and contingencies. Chapter 3 describes a SCADA system and its components. Chapter 4 describes the case study of the "Halifax water plant" where we examine contingencies in Sequential flowchart of "Coarse Screen" algorithm and handle contingencies using the resumption and termination and exception handling techniques. The chapter 5 offers summary and conclusions on the case study and future work.

Chapter 2

Background and Literature Survey

2.1 Background

Contingencies are not exceptions but consequences of external world changes. A contingency is an unusual but anticipated situation. The characteristic of a contingency is that, when the external environment has changed, no previous state may be valid; as the world has moved on, previous consistency properties no longer hold [6]. We also need to look at how contingencies are interrelated with exceptions. Because exceptions and contingencies are particularly important for embedded and real-time systems, we first briefly describe them in the background section. This is then followed by a literature survey on exceptions, contingencies, and termination models. We have merely illustrated examples of contingencies and how they could be handled. We have not intended to address all possible exceptions and contingencies.

2.1.1 Software-Intensive Systems

Systems generally may involve people, hardware, and software. According to Rich Hilliard, "software-intensive systems are those complex systems where software contributes essential influences to the design, construction, deployment and evolution of the system as a whole"[7].

In terms of distributed control system, the software components of a software-intensive system deeply interact with non-software components of the physical world, which is the environment within which the system operates. Moreover, in "modeling (and analyzing) a SIS (Software-Intensive Systems), the central role of the environment constitutes the main concern of the software engineer" [8, p.1]. The majority of software-intensive systems are embedded [8].

2.1.2 Embedded Systems

"An embedded system encompasses the CPU as well as many input/output resources such as memory hierarchy, and a variety of interfaces that enable the system to measure, manipulate, and otherwise interact with the external environment" [9]. What matters most for the correctness of an embedded system is that the parameters or set of attributes in the model of the embedded system represent the physical system. The model must maintain accurate correspondence with the physical system. The parameters often represent real-world quantities such as location, orientation, velocity, acceleration, temperature, pressure, signal strength and available amount of consumables.

ACID (Atomicity, Consistency, Isolation and Durability) properties may interfere with real-time embedded systems, such as SCADA [10]. Atomicity is often referred to as "all or nothing" semantics, and implies that any values modified while attempting the transaction be rolled back to their previous values if the transaction fails to complete. The consistency property means that a transaction must transform a database from one consistent state to another one. Isolation means that the effect of executing one transaction must not affect the execution of other transactions. Durability is defined as the effect of the transaction will persist and the transaction will not be rolled back.

The real-world quantities such as pressure, temperature, acceleration can only be known inexactly and typically may not be constant. In real time systems it may be essential that the values and data structures internal to the computer correspond to the value of the external quantities, which are outside the control of the computer system. Some may change discontinuously; however, if they are continuous, they may be predictable only over brief intervals and even then only with limited accuracy. Many models need to be calibrated with measurements that are not part of the operational system. By definition, all models are simplifications. They are incomplete and, for computational reasons, often are at best approximations that fail to predict unanticipated behavior [10][11][12].

Embedded systems have sensors and actuators. Sensors enable the embedded system to be aware of conditions in the external world and, in particular, changes in

those conditions. Actuators enable the embedded system to attempt to affect conditions in the external world. Both are fallible; sensors sometimes lie and actuators sometimes are ineffectual - both can fail intermittently or permanently and, of course, situations can arise which are ambiguous and can only be resolved by sensors and actuators included in the design. Redundancy is a key consideration in establishing the embedded system's awareness of the physical system with which it interacts, resolving ambiguity and indicating equipment failure. Because the physical system acts under the influence of external environment change than just the actuators controlled by the embedded system, and because the effect of actuators may not be what was intended, sensors must monitor actuator effects.

In embedded systems, the integrity of computations internal to the embedded system (the objective of ACID properties) is secondary to ensuring sufficiently accurate correspondence between the model and the external world [13]. Basically, atomicity interferes with ensuring sufficiently accurate correspondence between the model and the external world because of the all-or-nothing criterion of atomicity. The "or nothing" choice may not be a legitimate option. The passage of time since the initial attempt that was initiated (passage of external "real" time has certainly happened) plus any additional changes in values of the external world which may have happened, meaning that no computation based on the original values can produce the results that should be expected given the external values that are current now. The changes occur during the failed attempt is that operation of sensors and actuators, and even the computer itself, uses up consumable resources such as fuel or battery charge. In the sense that the original values cannot produce the expected results, the "or nothing" choice is not legitimate. The nothing choice is inappropriate if any such attributes are relevant to the state. The basic argument of atomicity is invalid.

Consistency means only valid data is accepted and retained. However, any constraint underlying a validity of data rule is either an assumption or an intention - and (subject to the unreliability of sensors) observed data is what it is. If the observed data does not satisfy the constraint, it is the rule that has failed, not the data. Moving on to isolation, it is really a property intended to limit what can be performed concurrently. However, in the external world things naturally occur concurrently, and may not be independent. For example, the aforementioned consumable resources can

result in interactions between apparently independent operations. The transactions that are normally regarded as isolated, and treated by the software, may not actually be subtle interactions. Consumable resources are an example of such a subtle interaction that is often ignored [13]. The exhaustion of a particular consumable resource can prevent the use of a sensor, actuator, or the computer itself. Finally durability is a property intended to ensure computer-initiated updates do not get lost. However, because the external world is affected by more than just computer-initiated updates, there is no way to ensure that[13].

Embedded systems are indeed typically dedicated, but not in terms of functionality. They are dedicated in that the hardware, including sensors and actuators, is only capable of maintaining an accurate correspondence with a particular class of system in the physical world [14].

2.1.3 Real Time Systems

Real time is a time as measured in the context of the computer system's external environment, not within the context of the program. Time is critical in real time systems. According to Stankovic et al., "A real time system is a system whose correctness depends not only on the logical results of a computation, but also on the time at which the results are produced"[15]. Many, but not all, real time systems are also embedded systems. Many, but not all, embedded systems are also real time systems.

Real-time systems are commonly characterized as being driven by deadlines on task execution times [16]. However, this is an inadequate and misleading over simplification notation. There are systems that have no specific deadlines, but for which jitter, the RMS (Root Mean Square) variation from regularity of a repeating event that should be precisely periodic is the issue in real time systems. Synchronization of independent signals is another example of real-time problems that may not involve deadlines. Intercepts represent a class of real-time systems where too early is as bad as too late: deadlines alone are inadequate. Moreover, intercepts typically have no rigid deadlines because accurate directions, locations, velocities and accelerations at precisely known times, together with the ability to schedule actions to occur at

specific times, may permit planning the interrupt to occur at almost arbitrary subsequent times. Real time can be intrinsic because the validity of external data expires after a well-defined interval, and should not be used in further computation. These cited examples illustrate only a few ways in which real time is intrinsic and essential [16, 17].

2.1.4 Distributed Control Systems

A distributed computing system (DCS) is a collection of autonomous computers communicating with each other to achieve a common goal. A distributed control system is used to gather or acquire data from various distributed processes and sends commands (controls) to these processes. The distributed control system is regarded as a device that issues all commands, gathers all data, stores some information, passes other information on to associated systems, and interfaces with the people who operate the process. A DCS is used to control the functioning of industrial processes like environmental control systems, traffic signals, water management systems, and oil refining plants [16–18].

2.2 Literature Survey

2.2.1 Exception

Goodenough[3, p.684] states that, "In essence, exceptions permit the user of an operation to extend an operation's domain (the set of inputs for which effects are defined) or its range (the effects obtained when certain inputs are processed). Exceptions permit a user to tailor an operation's results or effects to his particular purpose in using the operation. In short, exceptions serve to generalize operations, making them usable in a wider variety of contexts than would otherwise be the case" [3]. However an exception is totally arbitrary as Goodenough pointed out, exception handling can extend that narrow definition's range and domain of whatever the designer chose as the normal operation to yield a broader definition, that broader definition too is still arbitrary as to how broad it is. Choosing a narrow definition for an operation may make that operation simpler to understand or may make the implementation of that narrowly defined operation more efficient.

Van Ellen and Hasselbring [2] states common guidelines: "A common recommendation is to use exceptions only for specification violations, to declare them explicitly within the interface, and redeclare them within the interface of the caller if they have not been handled, and to adjust their abstraction to the current interface abstraction" [2].

In hardware design, there is a distinction between interrupts and exceptions. Both occur during the execution of a program and disrupt the normal flow of the program's instructions. However, because exceptions are perceived as being triggered by the instructions being executed whereas interrupts result from external events, most designers of hardware processors have chosen to distinguish between interrupts and exceptions. Interrupts must be serviced soon, but handling can be deferred until it is convenient, a behavior referred to as "masking" the interrupt.

On the other hand, in hardware exceptions (for instance page fault, divide by zero, bad address, illegal instructions) cannot be masked because subsequent instructions (or subsequent instructions within the same process or thread, in a multi-process or multi-thread environment) may depend on the results of the instruction triggering the exception, so no such subsequent instructions will be executed until the exception has been handled. Curiously, general purpose programming languages do not seem to have inherent support to mask or prioritize exceptions, although parts of the mechanism could and possibly should be deferred, such as recovering resources which are no longer needed.

2.2.2 Extended Exception - Contingencies Handling

The idea of contingency handling, rather than "errors" or "exceptions" is quite consistent with what is needed for real-time and embedded systems. The best way to handle real-time and contingency situations calls for forward recovery, i.e., resumption, rather than roll-back or termination. The main focus of Van Ellen[2] is on single-threaded applications. Van Ellen claims that it is not easy to declare contingencies at development time and handle them without specification violation. It is true that contingencies cannot completely be determined in specifications but they can be handled to some extent in the specification level.

The main drawback is that contingencies disclose implementation details, as there

is no abstraction. Van Ellen and Hasselbring have attempted to handle contingencies in a single thread; a top-down approach is used to handle contingencies and exceptions, and the "continue" keyword is used for resumption [2, 19]. There should be a method or structure to handle contingencies at a specification level, which is why contingencies descend from checked exception whereas errors or faults descend from unchecked exceptions.

Generally, every system has some constraints; thus, different types of contingencies can occur in a system. There should be the same structure to handle contingencies as with exceptions or errors. It is a flawed idea to ignore the possibility that an attempt to handle contingencies in a system and looked after when they actually occur in a system.

The termination model or the resumption model assumes that there have been some instructions from the normal flow of control executed before the exception was raised and the effect of these instructions must be corrected in the exception handling. Avoiding the contingencies can also lead to degradation of system performance, inconsistency, loss of time and cost effectiveness and maybe even loss of human life [4, 5].

2.2.3 Exception Types

Goodenough's[3] seminal paper on exception handling observed that exceptions permit the user to extend the operation's domain or the range of a program. Exceptions are generalized and are also used to serve different contexts of operations. Exception handlers are used to deal with runtime errors or the failure of an operation. Goodenough describes two types of exceptions - range failure and domain failure - and states that classification is an opportunity to augment the result with additional information that might be ignored in the normal case.

Range Exception

Range is the set of values possibly produced by an operation, domain is the set of values possibly accepted as input to an operation. Range exception deals with failure of output assertions for specific input or for any input. A range exception occurs when some output assertion is not satisfied. The author has given the example of a

parity error, such as when we are trying to read a record from a tape and a parity bit is not set. If effort is expended to deal with such an exception but the exception is not handled, then the expended work and time effort used to rectify that exception would be a type of range failure. The author describes the many ways to deal with such failures:

- The caller method has the ability to abort the operation, or the operation can also trigger an event if an exception occurs or aborts the functionality and undoes all the dependent operations.

- The caller method has the ability to terminate the operation and return some partial results so that the programmer can change the normal control flow of the program or deal with range failures to fix up the values of a variable based on the obtained partial results.

- The program has the ability to follow termination as well as resumption. Sometimes it is necessary to follow some process on the basis of partial results that are based on resumed values. For example, if a door jams before it is fully closed (open), it probably has to be opened (closed) again. To do so, however, requires knowing how far it got before jamming. Unless the door has sensors to measure how far it got, the only way to know this may be from the local variables recording the actuator activities. Yet for the termination model with all or nothing semantics, these are exactly the kind of partial results that are discarded in order to restore the initial values.

Domain Exception

"Domain exception is a somewhat different type of exception. It occurs when an operation's inputs fail to pass certain tests of acceptability" [3, p.2]. Domain exception deals with the limitation of an operation to accept input that could be valid if the operation were defined on a broader domain.

A simple example involves saving the results in an integer data type of an arithmetic operation, when the operand is divided by zero and the output is a "divides by zero" error. Another example of domain failure involves saving a value of a long variable to an array of integers, but it will not accept the value due to limited bytes

are allocated to different data types. The caller function must give the appropriate information about the failure to deal with domain failure so that the programmer can perform pre-checks or modify the set of input assertions to deal with domain exceptions.

Results Classification

Goodenough[3] states that "Result classification is a type of exception that leads naturally to the use of status variables or return codes (i.e. output parameters whose value designates the type of result produced); there is no need to resume the operation because a valid result has been produced already"[3]. Result classification is a type of exception that mainly occurs due to a status variable. Result classification augments the result by providing a classification for it, but the result itself is needed, so termination throwing away the result is wasteful, requiring redundant computation to recompute the result.

Monitoring

Goodenough[3] states that monitoring is the exception condition used to notify an invoker when some condition occurs. For example, we can perform a set of operations recursively by setting a variable timer to check the level of water in a tank is not equal to the expected limit value. If an exception occurs within the process, we can retain/resume the actual values of the other dependent variables from the recursive process. Thus, for instance, a binary search tree, we can get intermediate results from the recursion function without an unwinding process.

In short, the author described that exceptions and exceptions handling are interleaving actions that do not simply deal with errors; rather, these are levels of abstractions. If a program is not producing the expected results, we can change the normal control of program based on another set of instructions defined in an exception handler block, and make the decision based on the intermediate results, thereby saving the effort of partial computation using resumption.

2.2.4 Exception Handling Models

Goodenough[3] makes the case that there are indeed situations where the appropriate action for an exception handler is to give up on the computation that was being attempted and try something else. However, there are also situations where the appropriate action for the exception handler is to make changes in the environment and then resume the computation. The former is called the termination model, and the latter is called the resumption model.

Goodenough also argues that the exception handler needs to function in the context of the invoker of the operation experiencing the exception in order to have a broader perspective of what is the appropriate response to the exception. Meanwhile, the handler typically needs access to the state of execution at the point where the exception occurred, and especially to partial results within the operator experiencing the exception, in order to make appropriate repairs[3, 20, 21].

Termination Model

At the programming level, Java and other programming languages follow the termination model. Java provides strong features to handle exceptions with the help of try, catch, throw and finally blocks. The termination model suggests that when an exception occurs in a system, the exception handler should give up on the computation being attempted and try something else. This implies the previous state was the correct state. Throw provokes an exception in some other process or some higher block level.

In Java, the control is transfered to the catch block or finally if there is abrupt completion of expression. "The code that caused the exception is never resumed" [22]. But we redirect the control to some other instructions to retain the normal flow of program [22]. It is useful in real-time and distributed system to retain the value of a variable because there may be a chance that we can get the expected value of a variable that is changing periodically due to change in environmental situation. If it follows all or nothing semantics, these are exactly the kind of partial results that are discarded in order to restore the initial values. This is known as the rollback or termination model[6, 23].

Resumption Model

Resumption means disregarding past history, and moving forward to a state which is known to be safe and from which processing can continue. While this safe state depends on current values, it does not need to be any state previously experienced[24]. Let us take as an example a transportation system; if a traffic signal cannot function properly, we cannot cause all the traffic to return to their earlier locations. An automated system should take the real runtime values from its log and make a decision to reach a correct state of all variables. This is known as forward recovery or the resumption model. The resumption model is just an unanticipated procedure invocation, with access to all inherited values at the point where the exception was raised, but with control flow continuing as normal when the exception handler returns.

Coordinated Atomic Action Model

The essence of this model is that it rolls back to a previous valid state, upon exception as if the attempted operations had never been tried. Randell et al.[6] used the concept atomic action for handling exception. However, when the external environment has changed, no previous state may be valid, as the world has moved on previous consistency properties no longer hold[6]. The simplest termination model is just an unanticipated transfer to an in scope label, changing no values, and possibly losing dynamically allocated storage and other resources. Termination models, such as Java's try-catch mechanism or Coordinated Atomic Actions (CAA) attempt to avert resource loss and provide predictable execution state for and post exception handling by copying on entry all values that might change within the exception block, and restoring those entry values if the exception is raised. This saving and restoration can clearly be unbound time consuming if we consider dynamic data structures. External state, even just external storage, cannot generally be restored, so CAA are restricted not to have external effects.

2.2.5 Overview of Other Related Work

Regrettably, many programming languages that have been introduced since Goodenough's paper, from Clu (introduced in 1975) to Java (introduced in 1995) have

chosen to ignore Goodenough's recommendations. Moreover, the termination model is supported, while the resumption model is not supported at all. If any part of the transaction is performed, the transaction is performed in its entirety. This is often referred to as "all or nothing" semantics, and implies that any values modified while attempting the transaction be rolled back to their previous values if the transaction fails to complete.

Transactions can fail because of the transaction explicitly being aborted due to anomalies during processing, because of unanticipated situations arising during processing or because of a system crash. Moreover, the termination model uses only "all or nothing" semantics, which provides no access to the partial results computed at the point where the exception occurred [3].

According to Krischer et al. [20], handling exceptions affecting many threads or many processes in a multi-thread or multi-process program is difficult. The approach was used by Krischer et al. [20], to add new semantics to an existing syntax such as entry, monitor, and access. However, the approach complicates the semantics and the implementation, and makes temporal behavior even harder to predict and to understand. Krischer et al. [20] present evidence that syntax and semantics of general purpose programming languages for interacting threads or processes are overly ornate and need simplification.

Two methods for handling exceptions, contingencies and recovering the system are forward error recovery (resumption) and backward error recovery (termination). Gorbenko et al. [19] state that SOA (Service Oriented Architecture) uses resumption through loosely coupled services. If there are synchronization problems, there may be propagation delays in transferring messages and exceptions can rise. Gorbenko also illustrates that developers of WSDK should make some effort to reduce an exception propagation time used by different WSDK tool-kits [19, 20]. The main key hurdle is a synchronization problem with other systems in SOA. However, independent SOA web services kits can be improved. The use of SOA is essential in many situations, so avoiding it is not an option. This seems an uphill battle to revise the standard of web services kits because these kits are widely used in the real world and their large existing customer base, the designer would face challenges to modify web services toolkit to provide different functionality or semantics [19].

Romanovsky et al.[17] attempted to describe the different types of exception handling in an executed process. These include: 1) Action level time constraint, means that the system should trigger an event when a particular action is completed. 2) Value-based constraint, means that the system should meet the condition of a defined value of a variable. 3) Time-triggered-based constraint, means that the system should trigger an event when it reaches a defined time. We will apply these constraints in modifying the "Halifax Coarse Screen" SFC (sequential flowchart) algorithm.

Chapter 3

Domain: Halifax Water Plant Real Time and Distributed System

The coarse screen system is used to filter the impure or waste water in Halifax water plant. The screen is used more in the fall season when more hard material or sludge comes with water like leaves, sand and rocks. The waste water enters from one side and filtered water goes out from other side of "coarse screen plant area" as shown in Figure 4.1. There are two coarse screens; main coarse screen and sub coarse screen system (located underneath the main coarse screen) as shown in Figure 4.1. The coarse screen system works automatically. If the main coarse screen is not functioning properly for any reason such as a hardware fault or more weight on the main coarse screen then the sub coarse screen should get active automatically. If there is no need of sub coarse screen then sub coarse screen should stop automatically. We are using a Delta V process automation product provided by Emerson to control the SCADA system of the Halifax water plant.

The main control valve, which is an integral part of the coarse screen functionality, malfunctioned in 2009 when a control system failure caused a severe flood in the plant. The repair of the plant cost in excess of $11 million and several components, including the "Coarse Screen" system, needed to be replaced. The plant is a critical system and it is very important to handle exceptions and contingencies to prevent danger to the life of the system as well as to human beings. We modified the existing "coarse screen" algorithm in the Halifax water plant so that the system should make decisions on the basis of conditions or environment and follow both resumption and termination models.

An important observation about SCADA systems is that they are loosely coupled collections of separate software systems. Not only is there no concept of a single thread of control, but contingencies that occur, affecting computations that may be in progress, are not necessarily triggered by those computations.

3.1 SCADA System

A computer system sensing and controlling some large physical system and its environment is known generically as a SCADA system. This is abbreviated from "Supervisory Control and Data Acquisition" system. A SCADA system is a computer system that monitors and controls an industrial process, infrastructure and facility process. "The industrial processes include manufacturing, power generation and fabrication and refining processes. Infrastructure processes can be public or private, such as a water treatment and distribution, waste water collection or treatment process. Facility processes include airports, ships and space stations"[25].

Some SCADA systems focus on the data acquisition aspect and control applies to the behavior of the sensors only. Other SCADA systems use actuators that affect the physical system, such as steering radar antennae or changing the vibration characteristics of airframe skin. SCADA is a very versatile monitoring and control tool that is beneficial when the processes are complex with large number of I/O points, especially when these systems are spread over a wide geographical area [26].

3.2 Architecture of SCADA

A SCADA system typically consists of a communications network connecting multiple MTU (Master Terminal Unit), RTUs (Remote Terminal Units), and PLCs (Programmable Logic Controllers). MTU is the main system of SCADA architecture as it manages and controls all the operations done by PLCs, RTUs and field devices. MTU can also reach every point that is connected to it to effect changes.

The master-slave communication method is used by MTUs and other devices of a SCADA system. The main difference between MTU and RTU is that MTU is capable of initiating the communication whereas RTU cannot initiate a message. RTU can only send a message when ordered by MTU. The way of talking to each RTU and beginning the cycle in the process is called "scanning". "Polling" is the technique used to update the MTU data with data from the RTU[26].

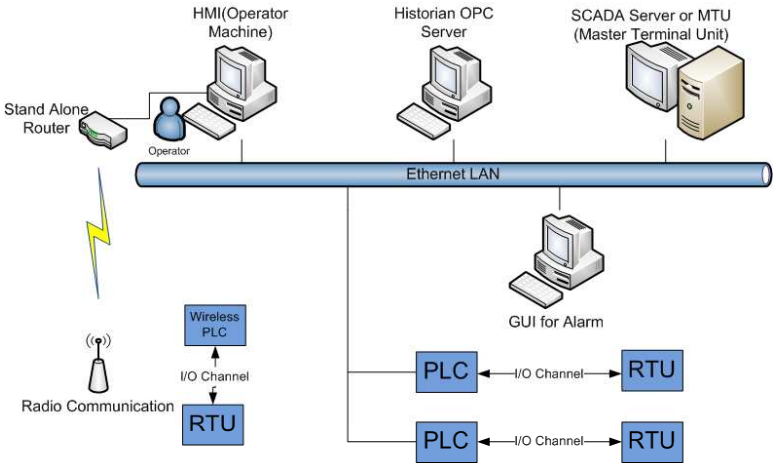

Figure 3.1: SCADA Architecture

As Figure 3.1 shows, the master terminal unit (MTU) is the heart of a SCADA system. MTU, human machine interface, historian OPC (OLE Process Control) server, GUI alarm and PLC devices are connected through Ethernet LAN or radio communication device. PLCs are connected with the remote field device through input/output channels. The wireless PLCs can get signals through radio communication.

3.3 Components of SCADA System

3.3.1 Human Machine Interface (HMI)

"The HMI (human machine interface) is intended to present process data to a human operator, with which he/she can control and monitor the processes" [26]. "Any popular GUI package, including Wonderware, Intellution, and RSView, can be used on HMI" [26]. Today's operator interface is likely to look like process flow diagram with 3-dimensional graphics, fully animated and interactive with the operator[26]. The graphics are just pictures and have no meaning until they are linked to the outside world. The GUI (Graphical User Interface) is configured to establish links between graphic elements on the screen. The calculations, analog or discrete values are pulled from RTUs or PLCs or other graphic elements.

The HMI has two types of links: 1) HMI actions and 2) HMI animations. The graphics with an action link, when actuated by the operator, result in some actions on the part of the control system. HMI has several types of actions such as navigational

links used to navigate through the screens, numerical data-entry links used to enter input parameters values, and local command links such as time scale on trend graphics display. Remote command links are used to send messages to field devices. The HMI animation link is widely used to show the actual processing of the system, as are visual components such as colors, images of SCADA devices and animations. For example, if the water level is high in a plant area, it will show that area in red, or if a system is in an idle state, it will show the plant area in a green color. HMI animation links include numeric analog value displays, status, graphics and alarm displays [26].

3.3.2 Programmable Logic Controller (PLC)

The programmable logic controller is used as a field device. It is more flexible and configurable than RTUs. A SCADA system consists of one or more PLCs sending messages to and receiving data from one or more RTUs. All data moved between MTU and RTU or PLC is in binary form. So, an A to D converter or a modem device is used for the communication. The data may be a status condition of an on-off switch or it may have been converted to binary from analog. The two types of PLCs are analog and digital. The digital PLC is used to monitor discrete signals.

In our case, we used boolean or discrete value for pressure and temperature on the basis of the operation. If the value of temperature or pressure exceeds a limited value, the digital PLC will send "true" values to the MTU. It depends on the designer of the device control; he can use either analog or discrete signaling for temperature or other control. The discrete inputs, like levels, pressures, temperatures, flows, valve positions, and motor status, can be monitored using simple switches. It is also possible to get the status of combustible gas alarms out of limited pH values. An analog PLC is used to monitor analog signals. It is important and hard to monitor analog signals rather than binary signals because binary signals just send one value true or false, while for analog, specific numeric value is used to control the device. So, for instance, to calculate the proportion of chemicals in the water, an analog signal is required[26]. Examples of analog signals are speed of motor, actual height of liquid or water in tank, strength of chemical component, and sensor values such as current. Suppose MTU instructed RTU to open valve VA-002 to 50% of its full operating range. This would be done by an analog signal, depending on what type of signal the designer

chooses to appoint, whether analog or discrete, in controlling the field devices. The output from the RTU could control the valve, motor speed and other parameters that could be used to describe analog values[26].

3.3.3 Supervisory Control System (MTU)

This system is used to gather or acquire data from various processes and send commands (controls) to these processes. The center of each SCADA system is a device that issues all commands, gathers all data, stores some information, passes other information on to associated systems, and interfaces with people who operate the processes. This is also known as Master Terminal Unit (MTU).

Some manufacturers call it "SCADA Server" or "Host". The communications in MTU are initiated by application software. It must also communicate with LCD (Liquid crystal display) displays used by the operators. It must also pass data to co-operating business computers, OPC (OLE Process Control) historians, and to HMI[26].

Most supervisory control systems are graphically configured from pre-existing software components and communication facilities in the system. In these, the sensor data and the actuator actions define the systems. Delta V is typical of programming facilities used for SCADA[21]. The contingency handling must be expressed using a graphical view of the system. When all remote devices are functioning normally, there is no need to assign an operator to inspect the remote locations frequently.

A safety control system should have main design characteristics, such as the ability to override when an exception occurs. Further, it should have independent components that should not affect the normal operation of the other components of the system. As well, it should be simple, easy to use and understand, have a well-defined structure, and the safety instruments should be able to be manually or automatically initiated by the control system.

The radio communication system used in SCADA at Halifax plant consists of one MTU and many RTUs. The MTU sends a message to its radio to start transmitting the signal; it then waits until the radio transmitter is ready to send a signal, modulates the transmitter with a message it wants to send to a particular RTU, turns the radio transmitter off, turns its receiver on, and waits for the RTU to answer. This is known

as MTU "turn-on time", a function of the radio that is generally independent of the data rate. Thus, reduction of scan interval has no impact on the data rate or on communication equipment[21].

3.3.4 Radio Communication

Figure 3.1 shows a radio communication system. Some SCADA systems cannot get a sufficiently short scan interval unless the data rate is pushed very high. When this situation exists, a medium with a bandwidth greater than a voice grade line will be required, such as optical fiber cable, microwave radio or a leased phone line on one or more sophisticated radio systems. "Bridge taps and load coils normally occurring on voice-grade lines can result in electrical pulses being slightly out of shape"[27]. Thus, radio or telephone cables have been developed specifically for SCADA. These offer flexibility, low cost, high reliability and better performance than normal wired system or expensive ultrasonic and supersonic waves. As well, radio signals provide a high wavelength over a small spectrum and require less frequency compared to high waves.

The two types of communications commonly used in SCADA systems are one-way and two-way exchange. In SCADA, data must move in both directions because telemetry systems gather information from remote sites but do not control the remote sites. For this reason, telemetry systems can use simplex communication, which allows messages to travel only one way. However SCADA requires two way (duplex) communications because it performs operations, monitors and controls[26].

The reliability and maintenance of the radio signal and equipment are the main issues with SCADA because, for instance, antennas loosened by wind must be realigned or the radio signal can be damaged by lightening. That is why control systems sometimes do not get a signal even on a very clear, calm and sunny day [26]. It could drift from its location and thus require checking and repair. Nevertheless, these communication systems are much better now than they were in past and are therefore more popular. Currently, they are used in industries because of their low cost and comparatively better performance.

3.3.5 Remote Terminal Units (RTU's)

The remote terminal units are used to connect sensor signals to digital data. An RTU gathers data from field devices in the form of analog values, alarm and metered amounts and status points. RTU keeps this data available until the MTU demands it through the PLC (Programmable Logic Controller). When the MTU instructs, it codes and transmits data to the PLC, opens and closes the valves, turns switches ON or OFF, outputs analog signals that may represent set points. The field devices are sensors, actuators and other devices in the process.

Certain control calculations require input data from more than one remote site. These control functions are most logically completed at the MTU. All data moved between the MTU and the RTU are serial data. A single string of binary characters are sent one after other, with parallel communication used for short paths. The OSI (Open System Interface) protocol is used in SCADA systems to communicate digital bits in serial format. The MTU, RTU, PLC are called "data terminal equipment" (DTE). Each has the ability to formulate a signal that contains data that must be sent and can decipher a receiver signal to extract its data. Some RTU or PLC protocols are emerging as standards in SCADA systems such as MODBUS, DNP (Distributed Network Protocol), ASCII and IEEE 60870 [26].

3.3.6 OPC (OLE Process Control) Historian Database

A historian computer machine is a part of SCADA. The Object Linking and Embedded (OLE) process control historian database is used for web-based Distributed Control Systems (DCS), storing log files such as control logs, alarm logs, operator logs. Any changes in the state of field components are stored in the log files so that DCS records can be maintained. The HMI also has a database to store operator records in a local machine; other significant configuration information is typically stored in the MTU.

3.4 Language: DELTA V Product of Emerson Process Management

The Delta V software product is provided by Emerson for SCADA [28]. It is basically used to program methods to control the functioning of an automated and distributed

system. Delta V can be successfully installed on different application processes such as hydroelectric stations, oil or gas production, gas pipelines, electric transmission systems, water systems [21, 28].

3.4.1 Subcomponents of language

The Delta V uses different sub languages such as functional blocks, sequential flowcharts, ladder logic, structured text and instruction lists. These languages are defined by IEC Standard 61131 [28].

Functional Block

Functional Block is a graphical language, part of IEC 61131, to describe the logic of the block diagram. It is used to program PLCs (Programmable Logic Control) in DCS. It has the same features as other languages such as object based and object oriented and has access to local/global variables and inheritance properties. Programmers can easily define the logic of complex structures by applying the output of one block as the input of another[28].

Sequential Flow Chart

Sequential Flow Chart is also a graphical language used to program PLCs. The sequences of steps are known as actions and are connected to each other by transition states known as logic conditions. It provides the facility to do multiple transitions in one step and is mostly used in DCS. A Sequential Flow Chart is used to program a PLC for running a sequence of processes[28].

Structured Text

The Structured Text provides the textual editor window similar to the IDE environment for coding PLCs. Global variables, iteration loops and conditional execution statements can be created and functional blocks defined at one place[28].

Ladder Logic

A Ladder Logic is a rule-based programming language used to program PLCs. It is a GUI-based circuit diagram of relay hardware components and operates in conjunction with HMI, MTU and PLC. The term "ladder" is based on the two vertical rails known as I/O terminals, and "rungs" or "rules" are the sequential ladders. If any one step of the ladder condition is true, then it will set at output 1, and a combination of bits produces one output for RTU. Each "rung" has one coil and a symbol at output side. These coils are regular coils (meaning, rung close), rung open, regulator contact, no contact. The limitation of ladder logic is that it is best to control the problems having binary input/output values. However, interlocking, binary sequencing and race conditions are the main problems with ladder logic. "Omorn", a manufacturer of various field devices, avoids these problems to some extent [29]. While analog values and arithmetic operations are not easy to define in ladder logic, the language is nonetheless useful for designing control programs for digital PLCs. The user of Delta V is expected to handle exceptions and contingencies through ladder logic.

The next chapter discussed contingencies in a SCADA system, case study of existing coarse screen algorithm and modified coarse screen algorithm.

Chapter 4

Case Study: SCADA for Halifax Water Plant

The functioning of the Halifax water plant is controlled through SCADA written in Delta V. Generally, a supervisory control system is used for managing typical signals that include alarms, status indications, analog values, and totalizer meter values. However, a great amount of information can be collected using the historian computer, such as ordering a motor to stop and resetting the "set point" values of valves. The Delta V product is easy to use and understand. The product Delta V by Emerson has GUI feature and a window for coding editor and also has an integrated development environment (IDE) to design electronic circuits.

Historian is a part of SCADA architecture. It is a computer machine used to save and retrieve backup information and the states of field devices. MTU must retain certain classes of data. A graphical form of analysis such as "trending" is another feature provided by Delta V. An operator can call on trend data plotted against time, such as easy-to-visualize data in the form of graphs and charts. Plotting two or more trends shows interdependence of sets of data that were otherwise difficult to visualize. An operator can create demand for MTU equipped with large data historian database to retrieve the stored values of field devices. Most data that is not critical for operations, such as historical information, should be stored in a central data store. The data that is vital, such as configuration information, should continue to be stored in the MTU. The information which is not worth retaining should be automatically erased after an expiration date is reached.

Figure 4.1 shows the "Coarse Screen" of the sequential flow chart algorithm. This screen is used to remove the sludge from the waste or storm water. The *HX-110-VA-001* is the main coarse screen device. Initially, water enters from the *HX-110-VA-001* door and the water flow takes a turn to the left and crosses to the *HX-110-XR-001* door. The main coarse screen filters the sludge from the water, lifts the sludge up to 80 feet, and pours it into the duty screen chamber. The filtered water crosses through

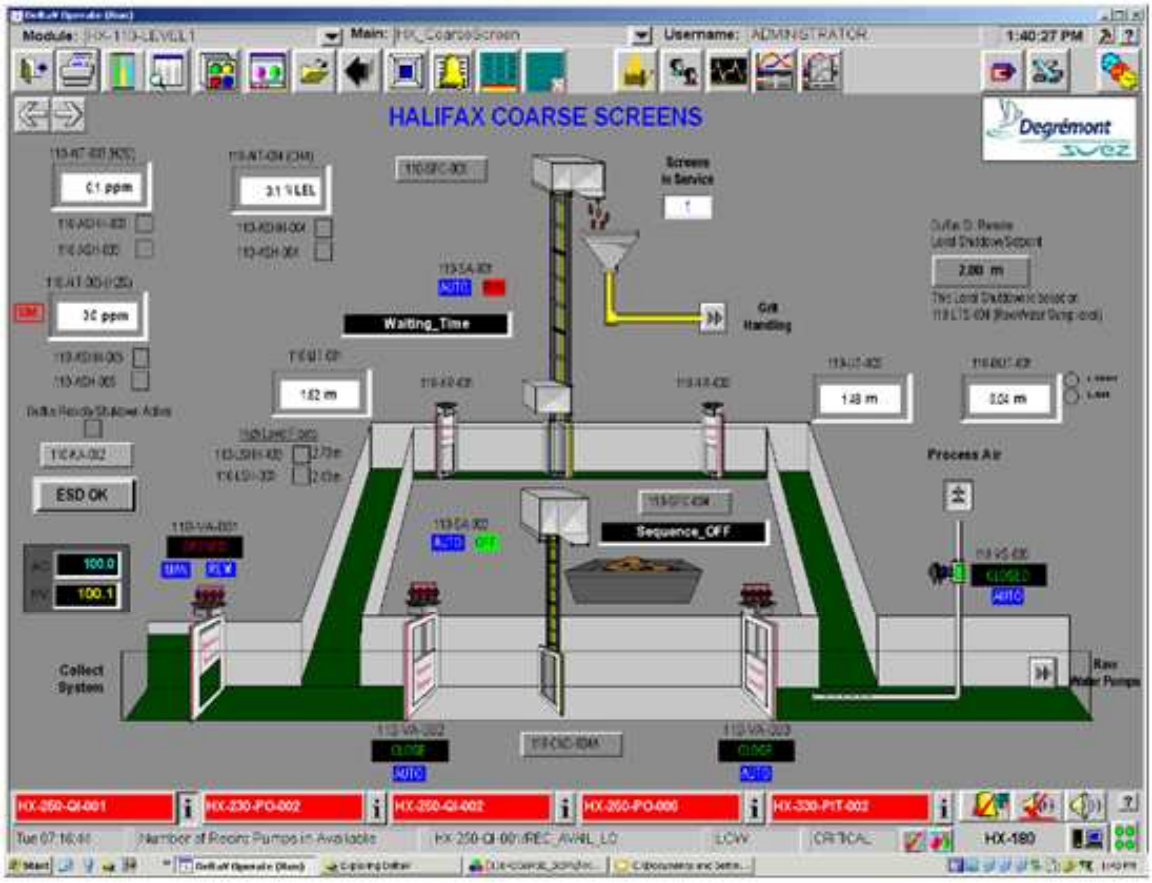

Figure 4.1: Halifax Coarse Screen[1]

the *110-XR-003* door and is pumped up by water pumps.

All of these valves and doors work as control devices. If there are excessive loads of sludge on the coarse screen or some other conditions such as level of safety, differential levels, or the main coarse screen is not functioning properly for some other reason, the sub coarse screen should start functioning. When the sub coarse screen starts functioning, *HX-110-VA-002* and *HX-110-VA-003* must be open so that the flow of water can move in its direction.

4.1 Proposed Flowchart

Using Goodenough's model, which concerns exceptions and contingencies handling in real time and distributed systems, we modified the sequential flowchart of the "Halifax Coarse Screen" system for handling exceptions and contingencies in real time

[1]Halifax Water Plant, Halifax NS

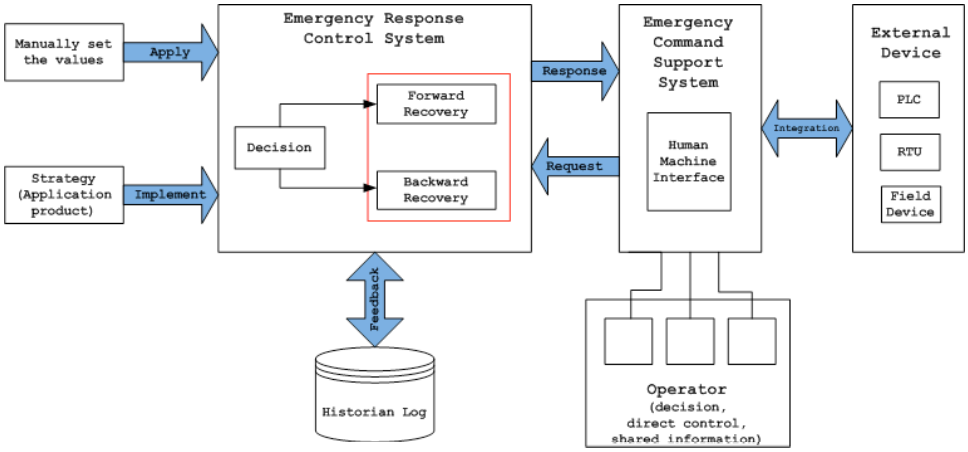

Figure 4.2: Resumption and Termination Model

and distributed systems. The Supervisory Control and Data Acquisition system has components such as historian log, master terminal unit, programmable logic control, remote terminal unit. In Figure 4.2, master terminal unit is a emergency response control system which takes the decision on the basis of resumption and termination models.

The operator can communicate through the master terminal unit via human machine interface, also known as emergency command support system. The operator can control the functioning of distributed control system manually and remotely by using product application strategy. The operator can send and receive the control information to field devices such as remote terminal unit (RTU), programmable logic control (PLC) and save the information to log files and backup information in historian computer machine.

According to Romanovsky et al.[6], in a real-time system design, recovery from errors and exceptions is divided into three categories by applying action level timing constraints, time-triggered coordinated atomic actions, and time-dependent exception handling. We applied value-based, action level timing constraints and time-triggered action for tolerating faults and handling exceptions and contingencies in real time system. We will identify where we have used each of these three categories.

We modified the sequential flowchart algorithm and applied resumption and termination techniques for handling contingencies on infrastructural real time application on the Halifax plant that is based on the model defined in Figure 4.2.

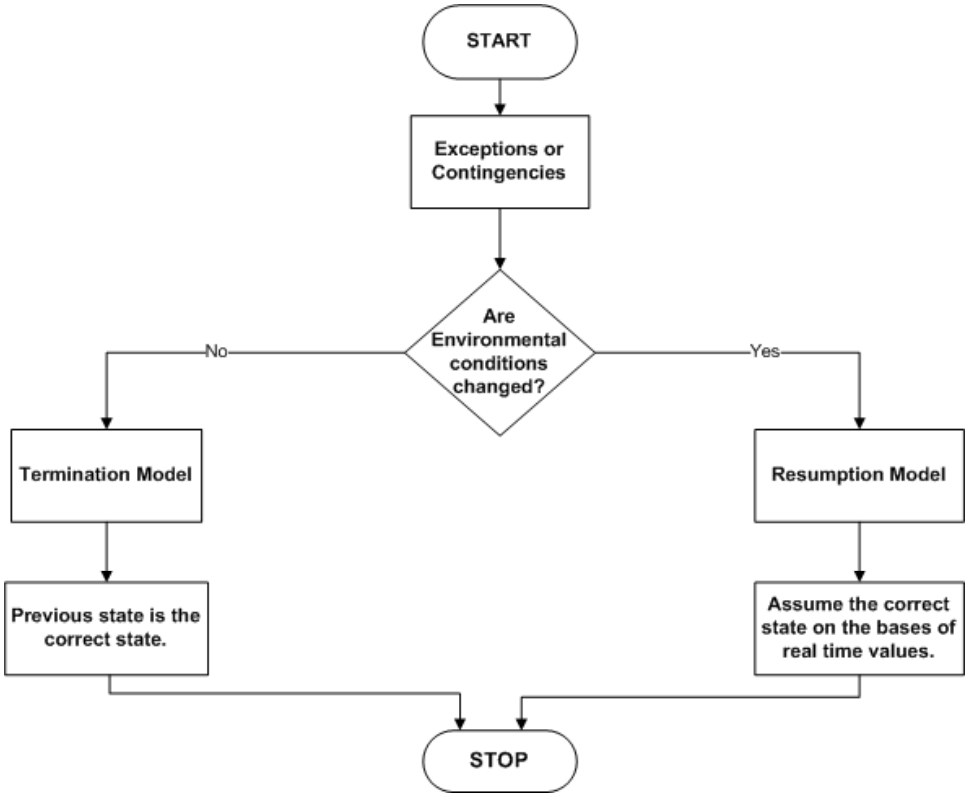

Figure 4.3: Proposed Flowchart

Although we have an object-oriented paradigm and structuring framework, the real-time aspect and behavior of the system adds more complexity in the design of this paradigm. That is why it is a challenging technique to develop an exception handling approach that can efficiently work in structural object-oriented, distributed and real time environments.

Figure 4.3 shows the proposed flowchart. If exceptions or contingencies occur in the system, the system should make the decision based on the environment change. If the environment is changed, we should not restore it to the previous values but follow the resumption model otherwise follow the termination model.

Forward error recovery (Resumption Model) needs to drive the system from its current state to one of the safe states. We applied the resumption and termination techniques on sequential flowchart algorithm of Halifax water plant.

The product Delta V provided by the Emerson has lack of programming constructs to handle the external world situations. As in general purpose programming languages used some programming construct to resume the correct state when any error occurs

but in Delta V, whenever the exception occur or when the system is not behaving normally, the designer can use the fail block to start the processing again. But we are checking more conditions to verify the call to fail state is correct or not by comparing the actual change status of the field effected by the external environment with what status should be expected by the logic mentioned in engine. So, we matched the actual and target state of the field devices and if it matches then further functionality will be perform otherwise rejected.

The actual state is the current state of the field device. The target state is collected based on facts collected due to environmental change and partial results. We use Fail block structure defined in Delta V and checked the status of each device by taking input into a fail block: if it passes this stage then the operator can manually operate the system, otherwise not.

We demonstrated the external environment change with the help of sensor values and the fail state sent by the field devices and handle the contingencies. We attempted to handle the contingencies to resume the correct state and redirect the control to normal flow. If the system is not able to reach to normal state, we used termination model to retain the previous correct state of the system.

It was convincing to the Halifax Water Plant staff to handle contingencies and that illustrated to them how they could handle other contingencies because contingencies and exceptions can occur in very obscure and unimaginable ways. In doing so, we modified the sequential flow chart of the "Halifax Course Screen" system for exceptions and contingencies. The backup coarse screen was tested through all possible combinations of values of the parameters that cause it to activate. Plus, the system was evaluated on how it handles exception and contingency situations in the system and how well it recovers from those conditions.

4.2 Coarse Screen Flow Chart

Figure 4.4 shows the processing steps of the coarse screen. The function and the control of the operation of "coarse screen" of water plant in Figure 4.1 are done using this flowchart.

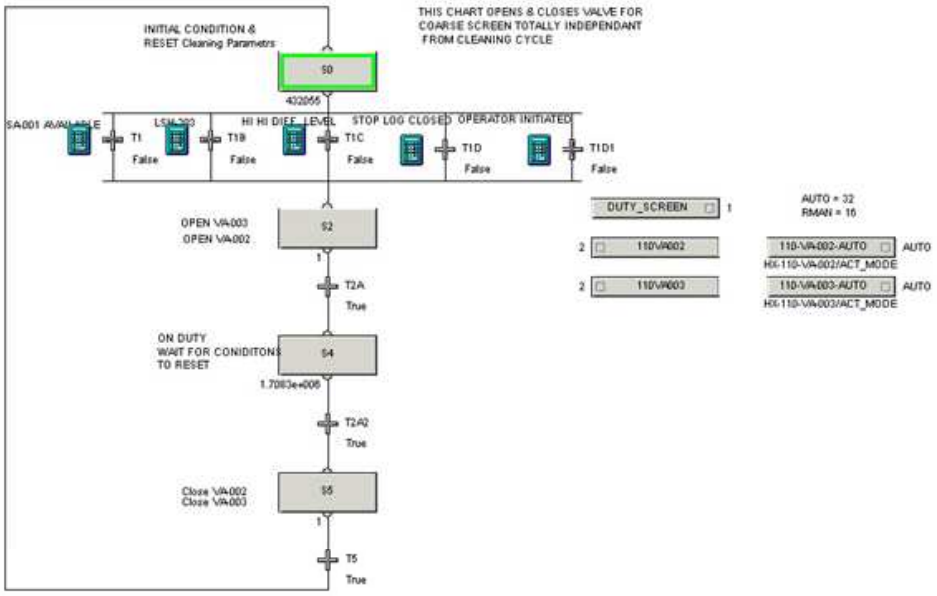

Figure 4.4: Coarse Screen Sequential Flowchart

4.2.1 Coarse Screen Sequential Flowchart (SFC) Algorithm

The current SFC algorithm is partially automated and contingencies are not handled in this system. Following are the steps of the previous SFC algorithm:

Step 1:

Set OPEN button to 1 and check Initial conditions:

Read the status of DUTY_SCREEN, VA-002, VA-003

if *((DUTY_SCREEN == 1) and (VA-003 == 32 and VA-003 == 0) and (VA-002 == 32 and VA-002 == 0))* **then**

  go to next step S0 in Fig. 4.4 to check pre-conditions;

end

Step 2:

Check pre-conditions of T1, T1B, T1C, T1D, T1D1:

T1: SA-001/CND/Out.D.CV =1(SA-001 is unavailable)

T1B: LSH-303 is high

T1C: High - High Differential Level

T1D: stop log == closed.

T1D1: Operator Initiated.

if *((T1 or T1B or T1C or T1D or T1D1) == 1)* **then**
  go to next step S2 in Fig. 4.4 to open the valves VA-002 and VA-003;
else
  go to next step S0 to check pre-conditions;
end

Step 3:

Set VA -003 to 1 and VA-002 to 1 **if** *(VA -003 ==1 and VA-002 ==1)* **then**
  go to next step S4 in Fig. 4.4;
else
  wait for opening the valves at this step;
end

Step 4:

T2A: Check the condition either valves are open in AUTO or REM mode

if *((VA-003 == 1 and VA-002 ==1) or*
((VA-003 == 1 and VA-003 ==16) and
(VA-002== 1 and VA-002==16))(Remote manual)) **then**
  go to next step to check the post conditions;
end

Step 5:

T2A2: Post Condition: (T1 and T1B and T1C and T1D and T1D1) is false, close the sub coarse screen

if *(T1 and T1B and T1C and T1D and T1D1 == 0)* **then**
  CLOSE_BUTTON = 1 go to next step S5 in Fig.4.4;
end

Step 6:

T5: To set the state of valves in auto mode

if *(CLOSE_BUTTON == 1)* **then**
  set VA-003 to 32(AUTO) and VA-003 to 0(CLOSE)
  set VA-002 to 32(AUTO) and VA-002 to 0(CLOSE);
end

Algorithm 1: Sequential Flow Chart

(Here, 1 = OPEN or ON, 0= CLOSE or OFF)

4.3 Contingencies in Coarse Screen Algorithm

As we identified that the damage to the plant occurred due to the deficiencies in the SFC algorithm, shown in 4.4, we have concentrated on identifying and handling exceptions and contingencies in it. As it is not possible, even for this isolated algorithm, to identify all possible contingencies arising due to the real-time environment and the many parameters affecting the plant, we concentrated on those situations with the corresponding parameters that affect the function of the algorithm and the (sub)system it controls. Thus we have identified the following contingencies that affect the algorithm and cause the modification of the algorithm which will be provided later:

1. What if exceptions occur in an exception handler block?

2. If maintenance work is going on at the main coarse screen or it is not functioning properly and *XR-001, XR-002* are opened, then water will pass out from the system without being filtered. In this case, *XR-001, XR-002* should be closed and *VA-002 and VA-003* should be opened automatically.

3. If the operator shuts the doors *VA-002 and VA-003* manually, and breaks the functioning sequence of the chart and these valves, *VA-002 and VA-003* are closed and in REM (Remote Manual) state, then the initial conditions are not satisfied. The "coarse screen" will not work well the following day. It may or may not get stuck at any step of the sequence. This kind of contingency usually occurs when the operator breaks the sequence of flowchart.

4. If there is more sludge in the water, then high differential values or levels of safety conditions may occur. There may be a high differential value because the coarse screen has to lift it up 80 feet, which means it takes time. This may result in a high differential and/or a high level of safety concern. High differential occurs if the level of water in the "water plant area" is more to the left side in front of the filter "coarse screen" rather than to its right side behind the filter. The high level of safety refers to the water level being more than 2.0 meters high in the plant area, as shown in Figure 4.1.

5. If any of the initial conditions described in 4.4, SFC algorithm, are true, then *VA-002 and VA-003* should be opened automatically. However, what if the system did not get the feedback? Field devices such as actuators send a radio wave signal to the system.

   The radio wave signal may be lost due to a change in the direction of the radio wave's antenna or day-time and night-time lightening effects. Inclement weather is the main cause of not to getting feedback from the actuators and sensors or from any control hardware, even if the doors are opened. In this situation, it can fail or get stuck at the sequence of flowchart because there is no feedback status from the previous state. Hence, the condition to go to the next sequential step is not fulfilled.

   In handling the above described contingency conditions, the "coarse screen" system was modified by adding additional functionality, an ESD (Emergency Shutdown) button at the HMI (Human machine interface) screen to ensure the safety of the plant. When an abnormal situation occurs in the "coarse screen" due to an external environment change, the operator clicks on the ESD button. The structure of the plant has been saved by forcefully shutting down the *HX-110-VA-001.*

   The operator can click on the ESD if any part or the area of the coarse screen system is not functioning properly. After shutting down the whole system, an operator starts to diagnose the fault with the help of historical log information. The current status of each property value is displayed to the operator on the HMI screen and he/she inspects the area of the plant and rectify problems manually, such as forcefully setting the values of each controller at the software level and repairing at the hardware level.

   However, ESD is not the best way to handle exceptions and errors. It may also affect the functioning of backward pipes and valves as well as forward components such as Grit handling, the air processor and further connected air pumps.

4.4 Modified Coarse Screen SFC

Figure 4.5 shows the modified coarse screen SFC, fault tolerant system. This modified supervisory control system will automatically handle errors, exceptions and contingencies. The defined contingencies are not errors, as these may occur during the general operation of the system when the external environment changes and the system cannot perform the expected operation.

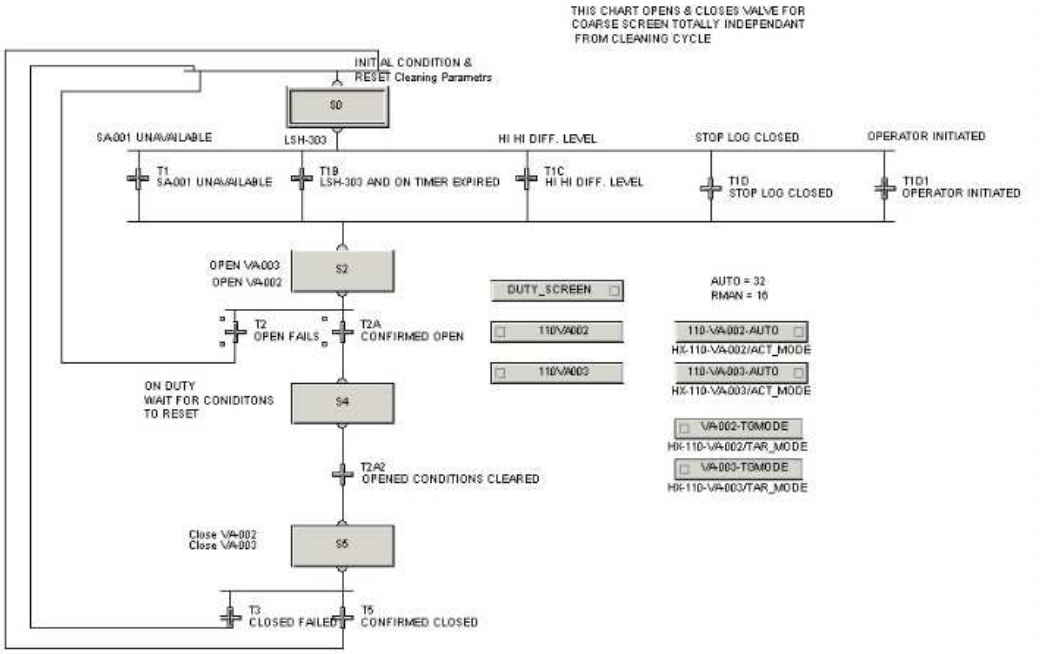

Figure 4.5: Modified Coarse Screen Algorithm

The operator has authority to set the system to MAN mode and reset the system when any exception occur. If the system is not functioning due to contingencies that occur in the coarse screen system as described above then then operator will have to contact the programmer to reset the logic forcefully. So, it may take one or two days to restart the coarse screen system. The advantage of this fully automated system is that it will save time, prevent faults, handle exceptions, errors and contingencies and no extra effort of manpower is required to diagnose the system. Further, as it will prevent flooding and breakage of the system, it will result in cost savings.

As Figure 4.5 shows, S0 to S4 are the sequences or steps connected with single or multiple transitions. Numeric number conventions are used to define the state of

physical devices. For example, the value of duty screen is "1" when it is active. For a coding perspective, we are using value "32" for AUTO mode, and "16" for remote manual mode. *VA-002 and VA-003* are in AUTO mode for initial conditions. In this modified flowchart, we are using target mode value of valves of *VA-002 and VA-003* to handle one contingency condition.

4.5 Modified Coarse Screen Algorithm

This algorithm is fully automated. The exceptions and contingencies defined in Figure 4.4 are handled in this system. Following are the steps of the modified SFC algorithm:

Step 1:

Set OPEN button to 1 and check Initial conditions:

Read the status of DUTY_SCREEN, VA-002, VA-003

if *((DUTY_SCREEN == 1) and*
((VA-003 == 32 and VA-003 == 0) and
(VA-002 == 32 and VA-002 == 0)) or
((VA-002-TGMODE.CV == 32 and VA-002== 0) and
(VA-003-TGMODE.CV == 32 and VA-003 == 0))) **then**
  go to next step S0 to check pre-conditions;
end

Step 2:

Check pre-conditions of T1, T1B, T1C, T1D, T1D1:

T1: SA-001/CND/Out.D.CV =1(SA-001 is unavailable)

T1B: LSH-303 is high

T1C: High - High Differential Level

T1D: stop log == closed.

T1D1: Operator Initiated.

if *((T1 or T1B or T1C or T1D or T1D1) == 1)* **then**
  go to next step S2 to open the valves VA-002 and VA-003;
else
  go to next step S0 to check pre-conditions;
end

Step 3:

S2: Open the valves: SET VA-003 to 1 and VA-002 to 1

if *(VA-003 == 1 and VA-002 == 1)* **then**
| go to next step;
else
| **if** *TIME.CV > 800* **then**
| | go to next step;
| **end**
end

Step 4:

T2A: Check the condition either valves are open in AUTO or REM mode

T2A.a: **if** *((VA-003 == 1 and VA-002 ==1) or ((VA-003 == 1 and VA-003 ==16) and (VA-002== 1 and VA-002==16))(Remote manual))* **then**
| go to next step S4 to check the post conditions
else
| **if** *(T2A.a == 1 and S4 == 0 and S2/TIME.CV > 800)* **then**
| | goto next step S4
| **else**
| | goto step S0;
| **end**
end

Step 5:

T2A2:

Post Condition: (T1 and T1B and T1C and T1D and T1D1) is false,

close the sub coarse screen

if *(T1 and T1B and T1C and T1D and T1D1 == 0)* **then**
| CLOSE_BUTTON = 1 go to next step S5
else
| **if** *(T2A2.a == 1 and CLOSE_BUTTON == 0 and S5/TIME.CV > 800)* **then**
| | goto next step S5
| **else**
| | goto step S0 ;
| **end**
end

Step 6:

T5: To set the state of valves in auto mode

if *(CLOSE_BUTTON == 1)* **then**
  set VA-003 to 32(AUTO) and VA-003 to 0(CLOSE)
  set VA-002 to 32(AUTO) and VA-002 to 0(CLOSE)
else
  if *((VA-003/DC1/PV_D.CV ==2 and VA-003/CLOSED/Out_D.CV == 16) or (VA-002/DC1/PV_D.CV ==2 and VA-002/CLOSED/Out_D.CV == 16) or VA-002/CLOSED/Out_D.CV == 1 or VA-003/CLOSED/Out_D.CV == 1)* **then**
    goto step S0;
  else
    T3: **if** *(S5 TIME.CV > 800)* **then**
      goto step S0;
    end
  end
end

Algorithm 2: Modified Sequential Flow Chart

(Here, 1 = OPEN or ON, 0= CLOSE or OFF, 16 = REM MODE, 2= REM MODE CLOSE, 32 = AUTO MODE)

4.5.1 Explanation

If an operator clicks on the open button and the initial conditions are true, such as a "duty screen" is ON meaning in active mode, then valves 02 and 03 should be closed and on AUTO mode. These are the initial conditions to start the coarse screen system. We applied value-based, action level timing constraints and time-triggered action for tolerating faults and handling exceptions in real time systems. When the "main coarse screen" is in functional mode, this SFC system will check the defined six conditions over a period of time, such as:

1. T1: If any system fault occurs in the main coarse screen or it is not functioning properly. This is a action level timing constraint.

2. T1B: If the level of safety is high. If its set point shows the level of water is high in the system area. This is a value based constraint.

3. T1C: If there is a high-high differential level, it means that the level of water is higher on the backward side of the coarse rather than on the forward side. This is calculated as:
   110-DLIT-001 = 110-LIT-002 - 110-LIT-001 If DLIT $>$ 1 meter; normal. If DILT $>$ 2 meter; high differential. If DLIT $>$ 3 meter; high-high differential.

4. T1D: If the stop log is closed, that means there is more load on the main coarse screen.

5. T1D1: If the operator gives the command manually to open the valves.

6. The main safe state is the initial condition in the described SFC. There is some safe state for each step. For example, at the second step, if any one of the five conditions is true and both valves are open, then it will proceed to the next step. In this case, there are only limited safe states in this chart, as described above: namely, T1, T1B, T1C, T1D, and T1D1. There are some other variables such as H_2S (Hydrogen sulphide), CH_4 (Methane) as shown in Figure 4.1 While these are immaterial (as they have no relationship with the contingency), there may be a dependency if the constrained value of these chemicals is high. In such a case, high pressured dangerous gasses are released, at which point everyone is prohibited from going into the plant area.

4.6 Contingencies Handling in Coarse Screen System

There is a possibility that when a new operator operates the system in "remote manual" mode, he will do so erroneously. Sometimes junk material can get stuck in the coarse screen that is not visible on the HMI screen. When the operator starts the system and it does not work, and if he is unable to detect any fault in the system, then he can set it to remote manual mode, forcefully set the values of variables, and attempt to open the door. However, there is a chance that costly instruments may be damaged by forcing them, which would be another type of contingency. We can analyze the contingencies conditions in an automated system when a system behaves

abnormally and does not produce expected results due to unexpected results. Some contingency conditions are critical and important to handle as they can result in large-scale damage, while others are normal and have a negligible effect on the processing of the system. Thus, it is up to the system designer to assign priorities to contingency conditions. For example, if a system motor is running continuously for long hours, the expensive instruments or motor may sustain damage due to high temperatures. It is up to the designer to define which contingency conditions have high or low severity. An example of high severity is that if the temperature of motor reached to its threshold value meaning that high temperature can damage the motor then running motor should stop functioning until the temperature of the motor is not reached to normal value or until the motor is cooled down. An example of a less severe condition is that the functioning of the coarse screen should not be prohibited even the motor is burnt out. It is up to the designer to define which contingency conditions should have higher or lower severity.

If any of the above-defined conditions are true, then the valves 002 and 003 should be opened automatically. However, sometimes the system does not get feedback due to loss of radio signals. Wireless field devices sends the response to MTU through radio wave signals, but sometimes the system does not get feedback due to some reasons such as loss of signal in case of radio communication and lose connection of wires in case of wire communication. Bad weather is the main reason for not getting feedback from the actuators, sensors or control hardware. Although the valves are open, they can fail or get stuck at a sequence in the flowchart because no feedback status from the previous state was received, thus the condition was not fulfilled to go to the next sequential step. In such as case, this condition is known as a contingency and is handled as if any of these above-described initial condition were true and time is more than 800 second then it should go to next the step. Generally, these valves or doors take 600 seconds to open or close. We applied time-based constraint. We are assuming the correct state value of those valves by applying resumption. Even if the sequences get stuck and are not able to proceed to the next step, the control is transferred to the termination state.

If all the initial five conditions are false, meaning that the main coarse screen is working fine, then there is no need to use the sub coarse screen. Valves VA-002 and

VA-003 should be closed when the main coarse screen is active and the sub coarse screen is inactive because the impure water should flow in the direction of the main coarse screen when filtering rather than toward the sub coarse screen. In this case, VA-002 and VA-003 valves should be closed automatically. The operator also has the authority to close these valves. In this, he has two options: either to send a "close" command through HMI or to set to remote manual mode and manually set the command to shut down the valves. The system is designed such that when the operator clicks on the "close" button of HMI screen, the system will first set valves status to auto mode and close it. However, according to the second option, when the operator forcefully shuts down, the valves retain the status value REM (remote manual). According to sequential flowchart, while starting the "coarse screen", the system checks the initial conditions first. One of the initial conditions states that the valves should be in auto mode and close. If the operator closes the valves using the second option via remote manual, the "coarse screen" system will not work next time due to the violation of pre-conditions. Moreover, when the operator's shift is over, a new operator comes on duty; if he shuts down the valves manually and does not press the close button, he will break the sequence of the flowchart. Thus, while the system might perform the CLOSE operation well at that time, when the operator starts the system again the following day, it will not work because the reset conditions are not fulfilled, meaning that *VA-002 and VA-003* are in REM mode and these have a status value of 1. This is another contingency. It can be handled in three ways:

- If an operator turns the system to ON, the system should ask the running hours value from the operator. When that time arrives, it should alarm the signal and automatically turn off the system. However, some operators are not experts and therefore could not predict how long the system should run, so we could not apply this technique.

- Another way is that every door or valve has an inbuilt controller and, with the help of log files, the system can assume the correct state of a particular valve and make a decision. We have handled this contingency here by placing the condition in Step 1 in algorithm 2. If the target mode status value of these two valves is 32, meaning that these valves are in manual mode and closed, it should check this condition, too, which indicates it is not an exception. This condition

may or may not occur during the normal functioning of a system. However, this condition does occur when the operator breaks the sequence of the flowchart and valves retain the value "16", which means remote manual. In this situation, the environmental status of the valves has been changed and initial conditions will not be fulfilled when the "coarse screen" is restarted. The operator should have the authority to manually operate these valves; we should not be able to prohibit them.

- Conditions at *HX-110-VA-001*: If the "set point" level value of the water flow rate is "0", then open the *HX110-VA-001* door fully. If its value is "1", meaning there is a high water level in the plant area, then open the door 50%. If the "set point" level value is "2", then close the door.

We have handled some contingencies, exceptions and errors at development time.

4.6.1 Functional Block Diagram of Initial Conditions

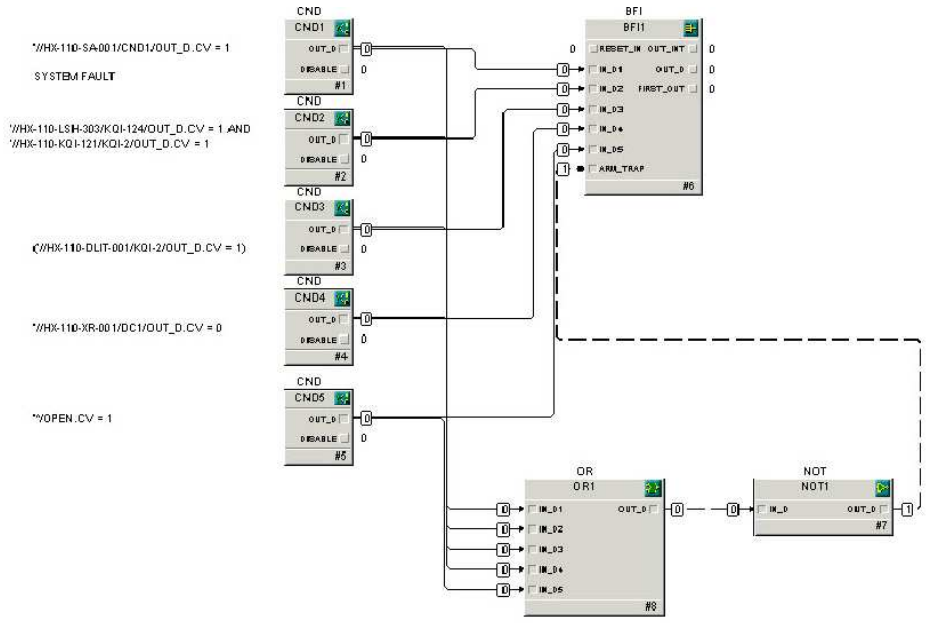

Figure 4.6: Functional Block Diagram of initial conditions

As shown in Figure 4.6, five condition blocks are used to take an input value from field devices to check five initial conditions as described in Figure 4.5. OR block is used to take input from condition (CND) block, and if any of the initial conditions

are true, then *VA-002 and VA-003* should be open. It passes the input value BFI (Boolean Flag Input) block and sends the signal to the output device such as PLC to control the system.

4.6.2 Functional Block Diagram of HX-110-VA-002, VA-003

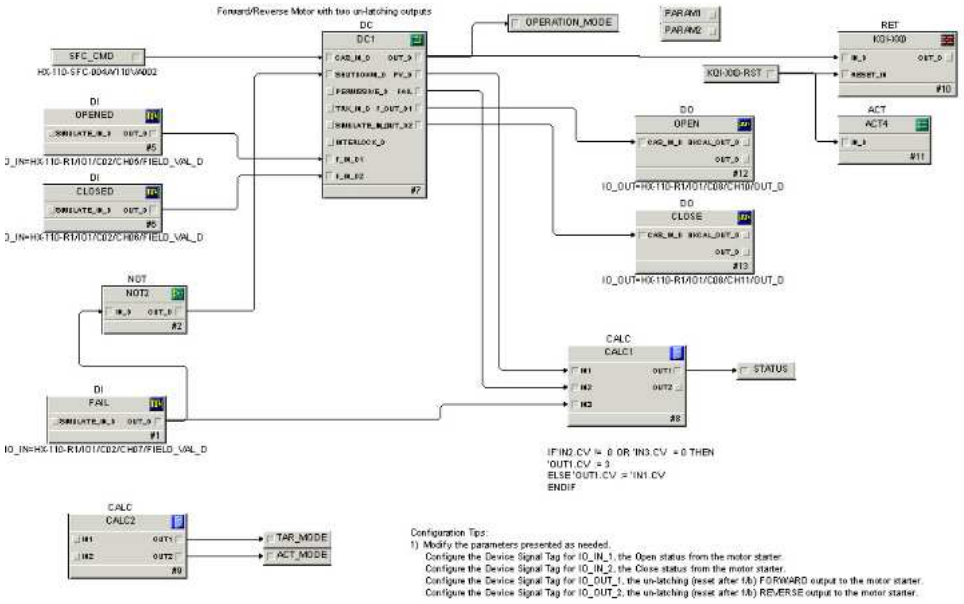

Figure 4.7: Functional Block Diagram of HX-110-VA-002

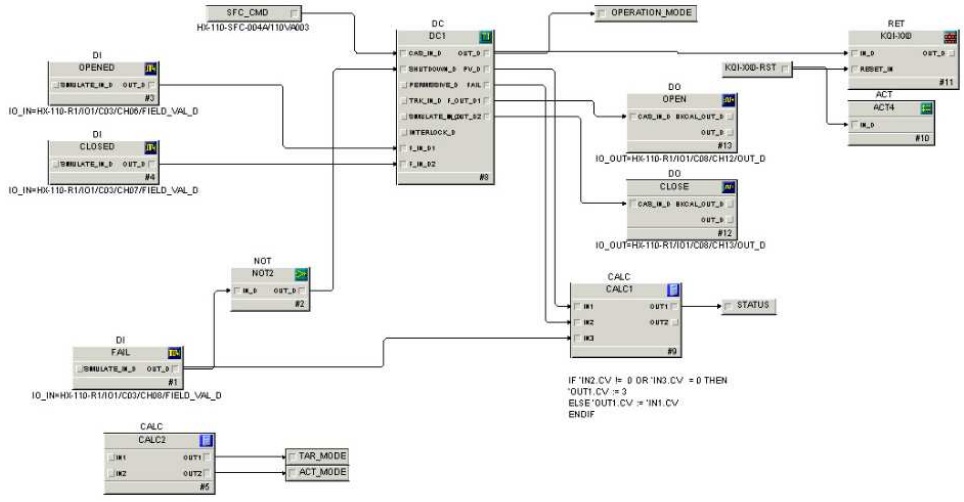

Figure 4.8: Functional Block Diagram of HX-110-VA-003

This functional block is detailed in Figure 4.6. Functional blocks also have an

inherited feature; thus, there is a possibility that when a new operator operates the system in "remote manual" mode, he may do so erroneously. Sometimes, junk material stuck on the coarse screen is not visible on the HMI screen. When the operator starts this system but it does not work, and he is unable to detect any fault in the system, he can set it to remote manual mode. In remote manual, he can forcefully set the values of variables and try to open the door. However, doing so may damage the costly instruments by forcing them. This can also be another type of contingency.

We can rectify this situation because each device is a 'smart device' with sensors, actuators, alarms, valves and thermostats. These devices have their own inbuilt control circuit and memory. Different types of failures can occur such as torque trip, configuration error, phase trip, thermostat trip, based on different alarms such as valves, control, actuator alarms, etc. If any error occurs in these devices such as valve fail, alarm fails and whatever is the cause of failure, a signal is sent to the RTUs. We added a "Fail block" to Figure 4.7 and 4.8 that will take this fail signal as an input and pass it to the DC block. This will stop the functioning of the system whether the system is in AUTO or in remote manual mode.

Furthermore, if the operator forcibly resets the conditions and clicks on OK, the system is protected and will not reset the values and trigger an exception such as valve alarm fail, causing torque trip, etc. The operator will then physically go into the plant and remove the stuck material from the coarse screen or cool the motor if its temperature is too high, and set the values of the valve through a remote device. The device will send an OK signal to the supervisory control machine and the operator can set or reset the values through HMI.

The resumption needs to drive the system from its current state to one of the safe states. We can handle this type of contingency by adding another safe state variable. Below are defined types of exceptions that are handled by this added "Fail state" block.

4.7 Fail Block Conditions

The added fail block in Figure 4.7 and 4.8 can take three types of values from sensors, actuators and other field devices. It is divided into three categories: valve alarm,

control alarms, and actuator alarms.

4.7.1 Valve Alarms

Torque Trip CL

When an operator clicks on the close button to shut the door VA-002 or VA-003, it will be tripped off because some material or object is stuck on coarse screen although this object is not visible on the HMI screen. If the operator forcefully sets the value to close the door, then the valve's alarm should trigger the "torque trip CL" exception signal.

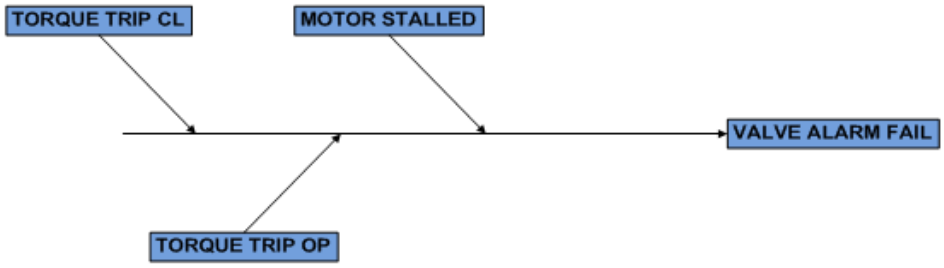

Figure 4.9: Valve Alarms

Torque Trip OP

Torque Trip OP is similar to "Torque Trip CL" but has an open operation rather than a close one. While moving the door VA-003 in the open direction, it is tripped off on torque due to some object stuck or by any other reason in the open direction.

Motor Stalled

When sending the request command to open VA-002 and VA-003 manually but motor is not started. So, it should send the fail signal to VA-002 and VA-003 controller circuit to stop the functioning of other dependent objects.

4.7.2 Control Alarms

ESD Active

If any exception occurs in the system area, 110-VA-001 will send a "control fail" signal so that it will prohibit the functioning of the system, even if the operator forcefully wants to start the system manually. The system will not work until the device sends the OK signal to "FAIL block" or to the supervisory control system. We have demonstrated and tested this functionality in a real time automated system at the "Halifax water plant". The "coarse screen" system ran successfully and handled both contingencies and exceptions.

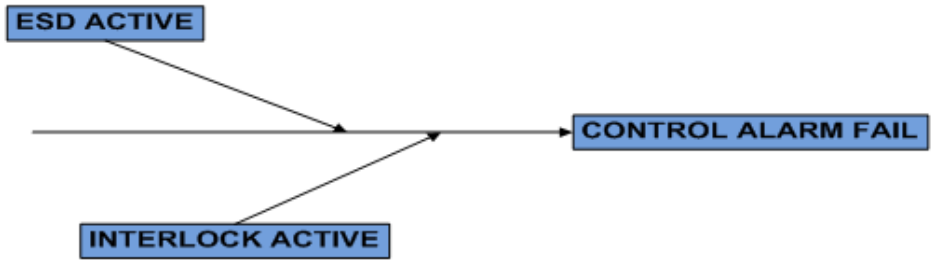

Figure 4.10: Control Alarms

Interlock Active

When the system is functioning and conditional control is configured, active interlock will not prohibit local control operation and will send a "control alarm fail" signal.

4.7.3 Actuator Alarms

Thermostat Trip

When a motor is overheated, it will automatically stop the motor until it cools down. It should send a "thermostat trip" signal as an "actuator alarm fail".

Phase Lost

The operation of the system is inhibited while phase supply to the actuator is lost due to any reason. It should send an "actuator alarm fail" signal.

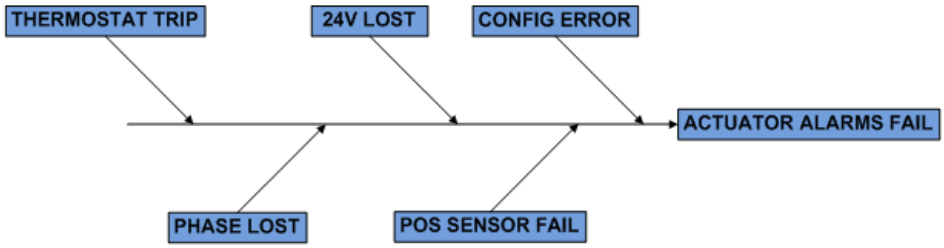

Figure 4.11: Actuator Alarms

24V Lost

Each field device has a permanent connection with a 24V supply. It resets the fuse and connection breaks from the 24V power supply automatically for the safety of expensive instruments. Sometimes, however, the connection can be lost naturally, even if there is no exception or abnormal situation occurring in the device. In this case, it should send an "actuator alarm fail signal" to the supervisory control system.

Config Error

When a programmer is configuring the actuator or setting the values, a contradictory condition can occur. In this situation, it should send a "config error" signal to the "FAIL STATE" block.

POS Sensor Fail

Sensors are not able to sense the signals due to change in their position. In this case, it should send a "POS sensor fail signal" to the Fail State block. The Fail State block will send the interrupt signal to the DC, which will inhibit the functioning of the system until these exceptions are removed or the OK signal is sent by the affected device to the fail state block.

4.8 Cause and Effect table of VA-002 and VA-003

The cause and effect table is currently the most popular format in industries to describe the functionality of field devices[30]. Such tables provide easier training to new operators, as they enable quick and simple analyzing. While Delta V does provide ladder logic to describe these tables, it is too elementary.

INITIAL	OPERATOR INITIATED	STOP LOG CLOSED	HI HI DIFF. LEVEL	HIGH LEVEL AFTER CLEAN	SA-001 UNAVAILABLE	CAUSE	110-SFC-004A	
						EFFECT		
						ACTION	TAG	DESCRIPTION
x						CLOSE	VA-002	Initially, VA-002 should be close because when main coarse is working fine, no need use this valve.
x						CLOSE	VA-003	Initially, VA-003 should be close because when main coarse is working fine, no need use this valve.
x		x	x	x	x	AUTO	VA-002	This valve should be in AUTO mode in all these described conditions except in Operator initiated case, this valve can be in AUTO or MAN mode in operator case.
x		x	x	x	x	AUTO	VA-003	This valve should be in AUTO mode in all these described conditions except in Operator initiated case, this valve can be in AUTO or MAN mode in operator case.
x						1 Screen	Duty Screen	Initially, The duty screen should be active means CLOSE mode.
	x	x	x	x	x	OPEN	VA-002	If SA-001 is not working or setpint value is 2 or high-high differential level, more load on main coarse or operator click on OPEN button, In these cases, OPEN this valve.
	x	x	x	x	x	OPEN	VA-003	If SA-001 is not working or setpint value is 2 or high-high differential level, more load on main coarse or operator click on OPEN button, In these cases, OPEN this valve.
		x	x	x	x	2 SCREEN	Duty Screen	The duty screen should be active means OPEN mode(ready to start work).

Table 4.1: Cause and Effect Table of VA-002 and VA-003

Siemens developed a tool "Simatic safety matrix" to design cause and effect tables. This tool is used for real-time monitoring, built-in event logs, simplified maintenance operations, self documenting and safety life cycle integration[30]. We have used two actions - open and close the valves - but it is easy to describe multiple actions with the help of a cause and effect table.

4.9 HMI Coarse Screen Controls

In Figure 4.12, the control screen is visible to the operator at the HMI (Human machine interface) computer. The "first out" is a smart alarm used to trigger exceptions.

It will trigger a signal if any of the five initial conditions are true. Some exceptions are not critical and have a low impact; an operator has an authority to bypass these controls even if the condition fails.

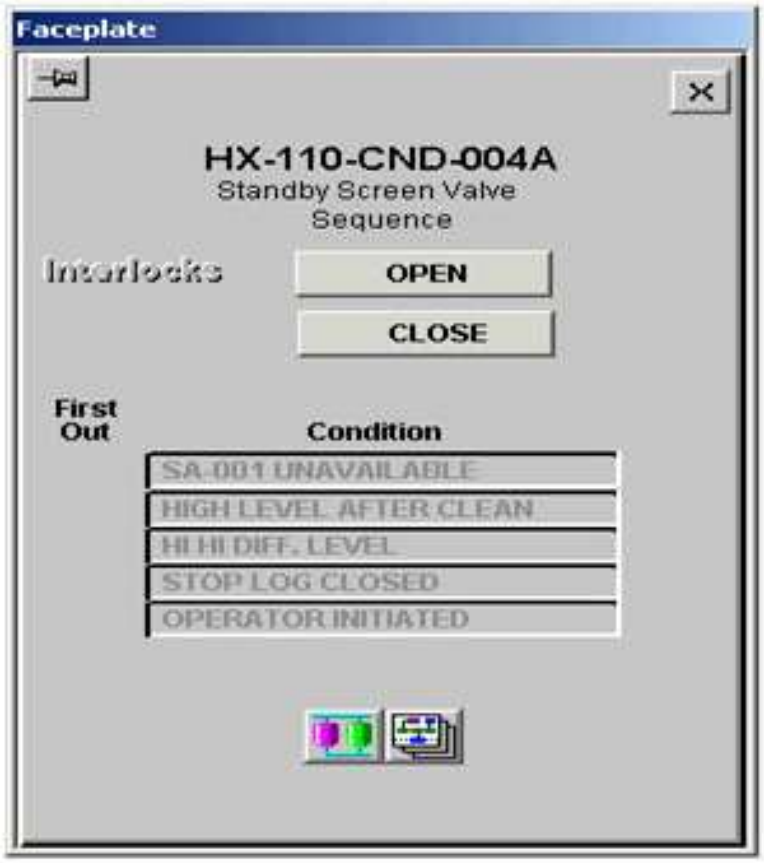

Figure 4.12: HMI Screen

The operator should not be authorized to bypass the conditions even it fails. The automated system should handle the exceptions and contingencies and redirect the control on other processes to recover it from a fail situation.

4.10 Transitions

These are the transitions of the updated flowchart, also known as condition blocks, for each step of the coarse screen algorithm.

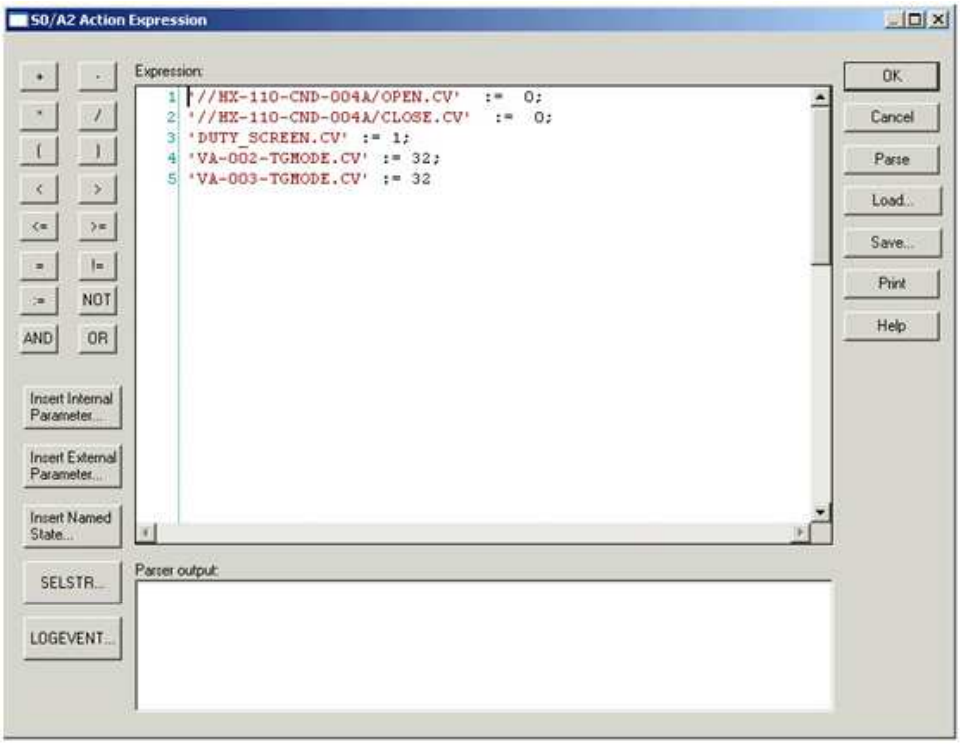

Figure 4.13: S0/A2 Action Transition

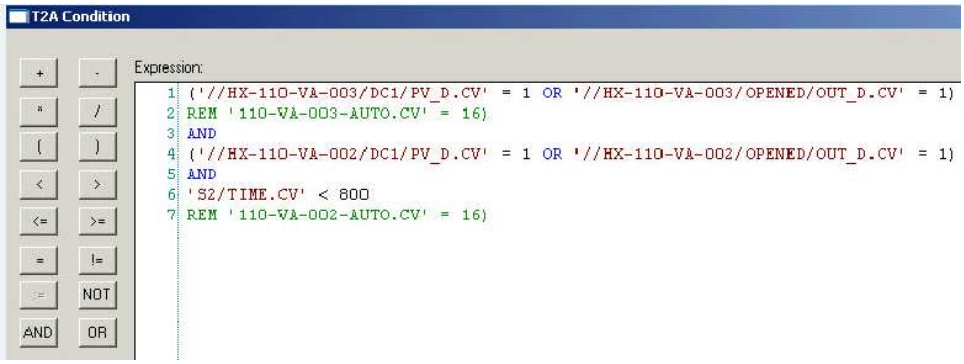

Figure 4.14: T2A Transition

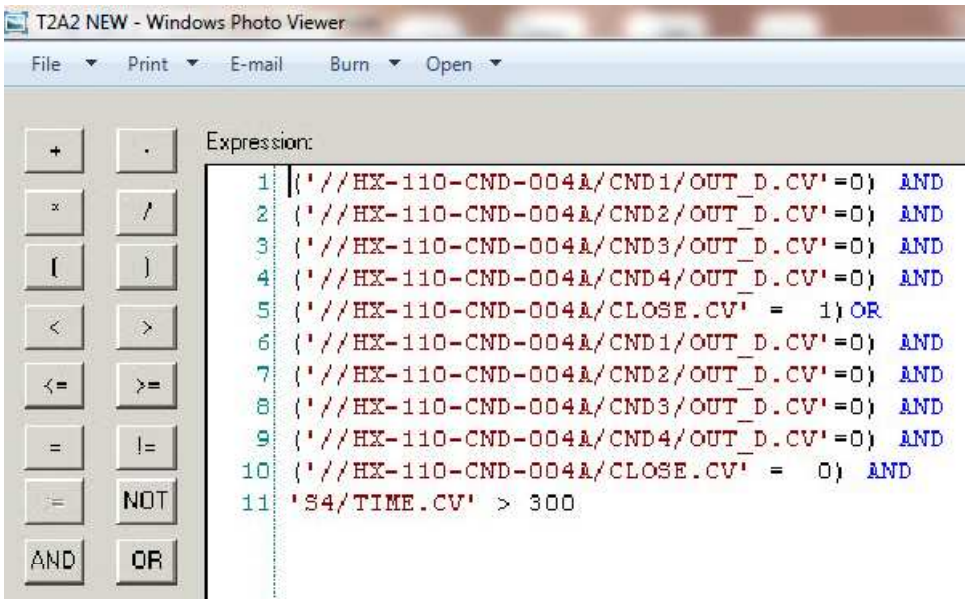

Figure 4.15: T2A2 Transition

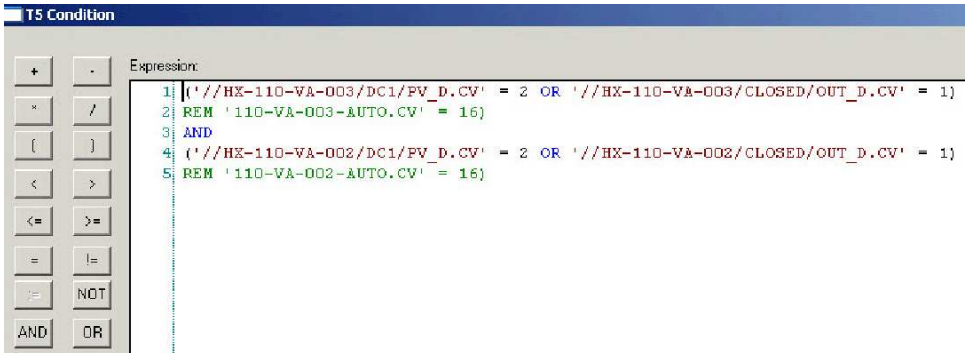

Figure 4.16: T5 Condition

Chapter 5

Conclusions

5.1 Conclusions

In this thesis, we attempted to handle exceptions and contingencies using termination and resumption models. A modification of the sequential flowchart of the "Halifax Coarse Screen", which is a real-time and distributed system, was devised to handle exceptions. This work was applied to an infrastructural real time application at the Halifax water plant. Real time and distributed systems are extremely complex, so it was not an easy task to handle exceptions, contingencies and tolerate faults in such systems. Because humans can operate SCADA systems erroneously, the system should be programmed to prevent self damage. It is imperative to handle exceptions and contingencies to prevent risk to the life of the system as well as to human beings. We modified the algorithm in the Halifax water plant, wherein the modifications enabled the algorithm it to make decisions based on conditions or environment and to follow both the resumption and termination models.

In certain situations, particularly in real-time systems, roll-back may be inappropriate to apply on a state variable. After an exception occurs, we cannot know the actual cause of the exception in Delta V because it has limited exception handling constructs. Instead, we have only partial results. In some cases, we applied both resumption and termination. The system will try to retain the normal state using resumption, but if the system fails to recover the normal stage, then it will apply backward recovery to maintain the safe state. De-allocation has some implicit runtime overhead. During that time, the programmer cannot do anything, so the resultant performance is degraded in time-critical distributed systems.

Our examination of the water plant identified a manual override that had been used incorrectly by operators, resulting in significant damage. Through the use of contingencies to identify the generic failure and then defining a forward recovery for handling the situation, we applied an automated recovery strategy that could cope

with system and manual errors but not permit operators to override safety rules.

Our principal contribution was that we found a way to produce an exception handling conditions to cope with the contingency that erroneous operator input could result in an invalid system state that might lead to damage. We handled the contingencies in a sequential flowchart of the "Halifax Coarse Screen" system. The backup coarse screen was tested through all possible combinations of values of the parameters that cause it to activate. Plus, the system was evaluated on how it would handle exception and contingency situations in the system and how well it recovered from those conditions. The system ran successfully and handled all defined contingencies and exceptions.

We assess the runtime cost of our solution not as a definite cost based on money, but a cost based on potential downtime and having the plant offline. If this back up system doesn't work when needed, the results could be a closure of the main plant gate that takes the plant offline. If the gate of the coarse screen system in Halifax water plant is closed for an extended period of time, the potential is that the collection system cannot handle the wastewater and overflow into the harbor or plant area. This goes against the plants permit to operate. The amount of effort was recorded as it involved DCS lead person in plant, plant supervisor and me and a few days of brainstorming to come up with the idea of how we can operate the fully automated coarse screen system efficiently. The DCS lead and the plant supervisor at the Halifax water plant were very happy and satisfied with this idea and solution.

Detection of the contingency need not be by conventional exception mechanisms, although that may or may not be the most efficient and highest performance implementation. The contingency such as junk in the water damming the flow (so that no water can reach a gate) or a motor burning out and failing (so that the gate cannot open or close) is a changed environmental condition. We can handle such contingencies using the termination or resumption model.

Delta V does not have exception handling facilities such as those in Java. The termination model or the resumption model assumes that there have been some instructions from the normal flow of control executed before the exception was raised, and the effect of these instructions must be corrected in the exception handling. By handling the contingencies at the specification level, however, we have detected the

contingency before any inappropriate instructions from the normal flow have been executed, and hence neither termination model nor resumption model recovery need to be followed to correct them. With forward recovery, once the contingency is detected it is merely necessary to transfer immediately to the safe state. Each automated system has an anticipated but unusual condition that occurs due to external environment changes. These situations can be detected and handled using the resumption and termination model so that an automated system should work efficiently and prevent any damage to itself.

5.2 Future Work

While designing web-based DCS, synchronization and loose coupling are the main issues in SOA (Service Oriented Architecture). Developers of WSDK should therefore make an effort to reduce exception propagation times required by different WSDK tool-kits. When a client sends a request to a server, the server performs all authorizations and other checks. Yet even when there are no pre-check conditions in a specific application, there is still a time delay. Pre-check conditions that are not necessary in a specific application should be reduced so there is a comparatively shorter time delay taken by the server to respond to the client. Additionally, the server can respond to the client with a specific message as to why the request was not completed, such as network connection failure, connection pooling delay, etc., rather than sending a general error message specified in WSDK. Krischer et al.[20] described 18 types of errors that can occur in a system.

Each distributed control system (DCS) has contingencies at specific levels. Exceptions and contingencies can be handled in other distributed computing systems, such as environment control systems, traffic signals, water management systems, oil refining plants, etc., but contingencies handling in multiple threads are the main issue.

DCS's main issues involve halting, asynchronous nature and mutual exclusion, as well as different speeds and configuration of I/O devices. These can be handled to some extent by using synchronizer and logical blocks and clock synchronization algorithms. Exceptions can occur in these devices and clock or devices can give inaccurate values.

Inconsistency is the main issue when exceptions occur in a distributed control

system, resulting in ACID properties forming the wrong base. Still, there is currently no widely accepted model that handles contingencies and exceptions in a real world environment. Sun Micro Systems and Microsoft could modify their toolkits to provide the same structure to handle both contingencies and exceptions. Likewise, the graphics featured in Delta V could also be improved.

In the Halifax water plant, the operator has to go to one GUI (graphical user interface) screen to set the values at different parameters and then switch to an HMI animation link screen to see the updated changes. In the future, it should be possible to provide a picture of the process just as it appears to the operator and to animate that picture to reflect process conditions simultaneously on one screen.

Bibliography

[1] Oracle Corporation. Definitions. http://download.oracle.com/javase/tutorial. Last accessed: Jan 5, 2011.

[2] Thorsten van Ellen and Wilhelm Hasselbring. Extended exceptions for contingencies and their implications for the engineering process. In *Proceedings of the 4th international workshop on exception handling*, pages 16–23. ACM, November 2008.

[3] John B. Goodenough. Exception handling: issues and a proposed notation. *Communication ACM*, 18:683–696, December 1975.

[4] Pat Chan, Michaelr Lyu, and Miroslaw Malek. Making services fault tolerant. In *Lecture Notes in Computer Science(LNCS 4328)*. Springer, 2006.

[5] J. Hansson, S. H. Son, J. A. Stankovic, and S. F. Andler. Dynamic transaction scheduling and reallocation in overloaded real-time database systems. In *Proceedings of the 5th International Conference on Real-Time Computing Systems and Applications*, page 293. IEEE Computer Society, 1998.

[6] J. Xu, A. Romanovsky, B. Randell, S.E. Mitchell, and A.J. Wellings. Coordinated exception handling in distributed object systems: from model to system implementation. In *Proceedings of the 18th IEEE International Conference on Distributed Computing Systems*, pages 12–21. IEEE Computer Society, 1998.

[7] Software Engineering Standards Committee. IEEE recommended practice for architectural description of software-intensive systems. Technical Report Technical Report IEEE Std 1471-2000, IEEE Computer Society, 2000.

[8] Carlo A. Furia, Matteo Rossi, and Dino Mandrioli. Modeling the environment in software-intensive systems. In *Proceedings of the International Workshop on Modeling in Software Engineering*, volume 11 of *MISE '07*, page 11. IEEE Computer Society, 2007.

[9] P. Koopman. Embedded system design issues (the rest of the story). In *In Proceedings of IEEE international Conference on Computer Design: VLSI Computers Processors*, pages 310–317. IEEE Computer Society, 1996.

[10] Jim Gray. The transaction concept: Virtues and limitations. In *Proceedings of the 7th International Conference on Very Large Databases*, pages 144–154, Sep 1981.

[11] Jie Xu, Alexander Romanovsky, Robert J. Stroud, Avelino F. Zorzo, Ercument Canver, and Friedrich Henke. Rigorous development of an embedded fault-tolerant system based on coordinated atomic actions. *IEEE Transactions on Computer*, 51:164–179, February 2002.

[12] Theo Haerder and Andreas Reuter. Principles of transaction-oriented database recovery. *ACM Computing Surveys*, 15:287–317, December 1983.

[13] Alexander Romanovsky. Coordinated atomic actions: how to remain acid in the modern world. *SIGSOFT Software Engineering Notes*, 26:66–68, March 2001.

[14] Rotork Controls Ltd.:Field Devices Manual. Sensors and actuators control. http://www.rotork.it/docs/1105/E170E3.pdf. Last accessed: Dec 27, 2010.

[15] Werner Schtz. Fundamental issues in testing distributed real-time systems. *Real-Time Systems*, 7:129–157, 1994.

[16] Ben Kao and Hector Garcia-Molina. Deadline assignment in a distributed soft real-time system. *IEEE Transactions on Parallel and Distributed Systems*, 8:1268–1274, 1997.

[17] A. Romanovsky, J. Xu, and B. Randell. Exception handling in object-oriented real-time distributed systems. *IEEE International Symposium on Object-oriented Real-time Distributed Computing*, ISORCP,8:32, 1998.

[18] Chao Cai, Zongyan Qiu, Hongli Yang, and Xiangpeng Zhao. Global-to-local approach to rigorously developing distributed system with exception handling. *Journal of Computer Science and Technology*, 24(2):238–249, March 2009.

[19] Anatoliy Gorbenko, Alexander Romanovsky, Vyacheslav Kharchenko, and Alexey Mikhaylichenko. Experimenting with exception propagation mechanisms in service-oriented architecture. In *Proceedings of the 4th international workshop on exception handling*, WEH '08, pages 1–7. ACM, 2008.

[20] Roy Krischer and Peter A. Buhr. Asynchronous exception propagation in blocked tasks. In *Proceedings of the 4th international workshop on exception handling*, WEH '08, pages 8–15. ACM, 2008.

[21] Emerson. Delta V online manual. http://www.emersonprocess.com. Last accessed: Jan 5, 2011.

[22] James Gosling, Bill Joy, Guy L. Steele, and Gilad Bracha. *The Java Language Specification*. Addison-Wesley, Upper Saddle River, NJ, 3 edition, 2005.

[23] David P. Pereira, Melo de, and C. V. Ana. Formalization of an architectural model for exception handling coordination based on ca action concepts. *Science Computer Program*, 75:333–349, May 2010.

[24] Y. Jayanta Singh and Suresh C. Mehrotra. An analysis of real time and distributed system under different priority policies. *World academy of Science, Engineering and technology 56*, pages 166–171, 2009.

[25] A. Daneels and W. Salter. What is scada? In *In Proceedings of the international Conference on Accelerator and Large Experimental Physics Control Systems*, pages 339–343, 1999.

[26] Stuart A. Boyer. *Supervisory Control and Data Acquisition.* ACM, ISA, 44th edition, 1993.

[27] Scott Mueller, Terry W. Ogletree, and Mark Edward Soper. *Upgrading and Repairing Networks (5th Edition).* Que Corporation, Indianapolis, IN, USA, 2006.

[28] Karl Heinz John and Michael Tiegelkamp. *IEC 61131-3: Programming Industrial Automation Systems Concepts and Programming Languages, Requirements for Programming Systems, Decision-Making Aids.* Springer Publishing Company, Incorporated, 1st edition, 2001.

[29] Omron Corporation. Products. http://www.ia.omron.com/. Last accessed: Jan 23, 2011.

[30] Siemens. Simantic safety matrix, preceded with confidence. http://www.sea.siemens.com, 2007. Last accessed: Jan 5, 2011.