http://www.the-control-freak.com

Building a Control System

by David Cass Tyler©

PO Box 1026
Willard, NM 87063
(505) 384-5342

David.Cass.Tyler@gmail.com

12/3/2010

Building a control system is most successfully done as an iterative process. Each successful step will help you to refine your understanding of the system, your goals and your desires for the system.

Table of Contents

1 Summary

Building a control system is most successfully done as an iterative process. Each successful step will help you to refine your understanding of the system, your goals and your desires for the system.

When you start the process, you generally know what it is that you want to do and how you want to do it. You will usually have some goals that look reasonable but are unrealizable because of time or cost constraints. During the implementation, you will undoubtedly realize that the system will be better if you do something differently. Once the system is running, you will recognize that the system itself has a natural rhythm or "sweet spot" that will cause it to run better.

Adapting to these changes in understanding and requirements will allow you to build the best system possible within your budget and time constraints.

2 Design

Careful design of the system is key to building a good control system. As in most things about computing, there is no "right" answer – just good and better. The design phase helps to focus you on building a system that satisfies the needs and goals that have led you to build a control system in the first place.

The problem with complex systems is the complexity itself. We need to design, with step-wise refinement, in a fashion that prevents obscuring our goals and requirements for the system. Simplicity is key to understanding the system.

2.1 Requirements Document

The Requirements Document is a statement of your goals for the system – i.e. what the system is required to do to be considered successful. It does not address any implementation details. It is simply a statement of the goals.

2.1.1 Summary

The summary description of the system should strive to be a single paragraph verbal description of the system being controlled:

"The Central Gas Facility will extract entrained hydrocarbons from the wellhead gas streams and inject them into the pipeline. It will also produce a miscible injectant to be used for secondary recovery."

If we cannot state the goals this simply, we probably don't have a good understanding of what we want to do, so we need to rethink what we are doing.

2.1.2 Description of the System

Give a brief overview of how the system will work:

"Incoming gas will be chilled to -45° Fahrenheit to liquefy all of the entrained hydrocarbons except methane, which will be diverted to the Central Compressor Plant for reinjection into the gas cap. Low end gases will be boiled off until the condensates have a low enough vapor pressure to be injected into the crude oil stream and sent to the terminal."

"Components of the system will include the chiller system, the stabilizer tower, and two liquid trains, each of which will include a de-ethanizer and a de-propanizer. Additionally, there will be a recycle line upstream of the de-propanizer to divert out of spec product back to the stabilizer tower."

"The low end gases will be run through two stages of compression to raise the pressure sufficiently to re-inject them back into the gas-cap."

2.1.3 Operational Requirements

This is a statement of the goals that must be met for the system to be considered a success. Even though the intent of this document is to describe the control system, operational parameters should be included as they affect the selection of control system components:

"The plant will process 7 billion cubic feet of gas per day and recover 135,000 barrels of condensates. The inlet stream will reach maximums of 165° Fahrenheit at up to 4500 psia and surges of up to 5 million cubic feet per minute. The outlet gas stream will be pressurized to 6000 psia and capability will be provided to send the entire outlet gas stream to flare for up to 15 minutes during an upset", etc.

Control system specific requirements go here:

"There will be an emergency shutdown button at every door of every module. Emergency Shutdown Procedures will be established to maximize the safety of the human operators and, secondarily, to minimize the damage to equipment. Operator Control Stations will exist in every module and will be capable of displaying current plant conditions for every control loop in every subsystem. Each will be capable of effecting localized control of that module in the event of loss of the main control system", etc.

Detailed specifications are deferred to future documents.

2.2 Functional Specification

Generally, the Functional Specification is a statement of what the proposed system is to do, leaving how the system is to be constructed to the design documents. In writing the functional specification, give only topical discussion of design considerations. The Functional Specification is a bridge between the Requirements Document and the Design Document. This is the preparation and guidance for the design.

Begin writing specifications to match each requirement. Leave the details of implementation to the design. Be certain that you've accounted for all requirements, both explicit from the customer and derived by us.

Give the Functional Specification a milestone in the Project Plan. During system development, the functional specification should be reviewed and updated periodically.

Use ample diagrams and examples throughout this document. This is especially true of wiring diagrams, data structures, and draft user interfaces.

2.2.1 What Do You Want the System to Do?

We have the generalized guidance of the Requirements Document to point us toward a stated goal. It is during this phase that we will describe how we will reach that goal. Start by listing all of the components that you can think of.

If we are going to build an High Power Microwave (HPM) Radio Frequency (RF) System, we will need a Power Supply, a Marx bank, a Pulse Compression Line and a Source for instance. The Power Supply will have to have a Power Source, the Marx bank will have to have a Trigger System (which will need a Gas Handling System), the Pulse Compression Line may need a Water Conditioning System, etc.

Complex systems can grow beyond our ability to understand them. We handle increasingly complex problems with increasingly simplifying abstractions. Subsystems are one of these abstractions.

2.2.1.1 What are the Subsystems?

Once you have a general overview of what you need, you can start grouping components by functionality. Arrange the components into a tree where each of the leaf nodes can function independently of the rest of the system. The key to defining a subsystem is to make it functionally independent of the rest of the system. It should include everything necessary to perform its stated functionality without requiring any knowledge of any higher functionality.

As an example, a gas handling system can be defined which is only responsible for maintaining gas pressure as measured by a transducer. This is a single control loop with a single process variable, a single set-point and a single control output. Multiple gas lines can be grouped into a single subsystem since they only differ by channel number.

You can then group subsystems into higher subsystems. For example, the trigger system needs a trigger generator and a gas handling system in order to assure that it can properly hold off voltage when required and then properly trigger a Marx bank on command.

2.2.1.2 Process and Instrumentation Diagrams (P&IDs)

P&IDs are easy to understand diagrams that show the process and it's control system in relationship to each other. The physical process is sketched out and the valves or

switches necessary to control it are shown and the transducers that provide the information on how the system is responding is drawn in.

These drawings give us a sanity check that we know what we are doing and can actually control the process – we haven't forgotten anything. In some cases, they will bring to light details that we hadn't previously thought of, so it can become an iterative process.

When these have been completed, we theoretically have a full list of all of the transducers and subsystems that will be required by the system. In fact, the next steps will probably add requirements to these, so we may be back.

2.2.2 What are the Projected Control Ranges?

Now that we generally know how we want to control the system, we can start specifying the actual pieces. We should have a good feel for "how many" and "what kind" of transducers we need. The ranges, including response times, will determine the selection of the actual components that we will use.

Remember to include the "upset" conditions when specifying ranges. We want to specify ranges that cover all of the values that we are ever likely to see. We may be planning to operate at a gas pressure of 60 to 120 psia, but we have to account for ranges of 0 to 150 or so psia to have full coverage of any gas pressure that we are ever likely to see.

We want to select transducers that will span these ranges without driving them to the rail (the minimum or maximum values a transducer will display). The selection of a transducer with a 0 to 10 volt output for a 0 to 150 psia full scale range is a bad choice if the pressure can ever go above 150 psia. Does 10 volts mean 150 psia or 200 psia? Because we have hit the rail, we don't know. A 0 to 160 psia full scale range might be ok since the pressure will never go negative. A 0 volt reading will always mean 0 psia, even if it is on the rail.

2.2.3 What are the Startup Defaults?

It is important to understand the state that a control system starts up in. Systems must be brought up safely to humans and equipment and in a controlled fashion where the system state is carefully regulated at all times. It is sometimes important to stage the times that equipment is turned on because the startup surges can overload the available power, etc.

It is in this (the Functional Specification) document that we initially specify the state required by each control output – which valves must be open and which valves must be closed for instance. We also specify the initial set-points of all of the control loops. We then specify the sequence that needs to occur for the system to be considered online and fully operational.

2.2.4 What are the Shutdown Defaults?

Equally important is how we shut the system down. There are usually separate requirements for "emergency shutdown" and "normal shutdown". Emergency Shutdown occurs when human life is at risk. Equipment safety takes a poor second in this case.

Normal Shutdown is the process of bringing the system down safely for both humans and equipment. In some cases, normal shutdown may include leaving parts of the system online (for instance, the vacuum subsystem may stay online and continue to maintain vacuum).

Failures are a consideration in the shutdown procedures. The meaning of the term "Fail Safe" is to implement the process of failing in a safe manner. For instance, the Safety Interlock System asserts a positive true so that a loss of power or a broken cable will assert false and the system will shutdown.

2.2.5 What Constitutes the "Safety Interlock" System?

The Safety Interlock System provides information about when it is safe to operate and when it is imperative to cease operations. Breaking the safety interlocks should be considered a danger to human life. The system should immediately put itself into the maximum safe state for humans.

Preference should be given to hardwired safety systems although various standards bodies have agreed to certain networked safety schemes. Hardwired has the advantage of working at the speed of light. By it's very nature it can bypass any susceptibility to collisions and stalls.

As an example, in our power conditioning system, which is capable of delivering a potentially fatal 300 Volts DC, the power to the solenoids is supplied by solid state relays that require a logic high signal. If the interlocks break, the power supply ceases and all of the solenoids automatically pop open and the dump circuits automatically close.

2.2.6 What Constitutes the "Emergency Shutdown" State?

Emergency shutdown immediately stops all activity that is potentially harmful to humans, turns off all high voltage production, disconnects the power supply and takes whatever other steps are necessary to make the system safe.

It then re-sequences the controllers to put them into the "Safe State" so that the operator unambiguously knows the current state of the system and equipment has maximum protection. Failure to return to the "Safe State" would leave the system in an indeterminate state and make it difficult for the operator to take any further action required to diagnose and/or repair the system before resuming normal operation.

2.2.7 What Constitutes the "Safe" State?

The "Safe State" is the initial starting state when the system is originally powered on. When the system is in the safe state, it is online but inactive. This is generally the same as the "Startup State". It is a state that is safe to both humans and to equipment.

2.3 Design Document

By the time we have progressed to the design document, we know both what we want to do and how we want to do it. It is time to get into the nitty gritty details. The Design Document provides the record of exactly how the system is to be built. The mandate for

the design document is accuracy and completeness – accuracy because inaccurate documentation is dangerous and worse than no documentation at all, and completeness because this document provides the final blueprint of how the system is to be built.

2.3.1 Fault Analysis

A Fault Analysis is run on the system to make the system as safe as possible in its final implementation. It provides an understanding of the worst case scenarios of operation. It provides the "what if" analysis of what could happen and provides the plan for how to handle these situations.

2.3.1.1 Detection

Especially for dangerous situations, you must be able to detect the problem. If you can't think of a way to detect this type of problem, sensors need to be added to allow detection. An example of this is failure for the dump switch to close. This situation would leave dangerously high voltages on the Marx switch. A good detection for this condition would be to use a solenoid with a micro-switch that would assert high when the solenoid was closed. Note that we would not use a micro-switch that would assert low as a power failure or a broken wire could give a false indication.

2.3.1.2 Remediation

Once you have detected the problem, you must have a plan for how to correct the problem. It is perfectly legal to put the system into "Safe" mode and holler for help. If the condition would preclude the safe mode or make it ambiguous whether or not the system was safe, you need an indicator, alarm or warning that alerts the operator of the potentially dangerous condition. An indicator for the "Safe" mode that failed to light could be used as an indicator of a dangerous condition. It is preferred to not use this type of "negative" indication.

The important thing is to think through, ahead of time, what needs to be done to protect people and property and to plan out a fix for every foreseeable problem.

2.3.2 Determine the Subsystems

You have identified all of the pieces and iterated on all of the problems and have a pretty complete understanding of what needs to go into the system.

Group the components into subsystems as discussed above. This is yet another iteration on that process, but we are getting close to the final version. Things have been added, and probably taken away, since the last time we did this. The subsystems and groupings that we choose at this point will forever effect the way that we think about and view the final system.

Take your time. Try combinations of subsystems. Remember that subsystems are one of our simplifying abstractions. We want to make the system as simple and easy to understand as possible.

Remember that we want an inverted tree. Each of the lower nodes needs all of the information necessary to control itself and those nodes below it. A node cannot depend on any information from its siblings or its parent nodes. It implements a stand alone system with all of the methods necessary to control that system. Parent nodes coordinate the transfer of information between siblings and they too follow the constraints listed above.

The final structure is very analogous to an organization chart. The workers at the lowest levels have all of the tools to do their jobs and each takes direction from above. Each management layer above them knows how to control its workers, all the way up to the president of the company. At each level, any loss of communication from above leaves the sub-tree still in control of its last commanded function. Loss of interlock causes every node to go to the safe state, so the system is always under control.

2.3.3 Describe the Endpoints

All of the information necessary to build, calibrate, diagnose and repair the individual components should be provided in this document.

2.3.4 Select the Transducers/Devices

Include the make and model number of the component. Everything necessary to unambiguously (re)order these components should be available. Datasheets and User Manuals should be available for each of these components.

2.3.5 Determine Ranges

Select ranges necessary to provide the resolution necessary and to stay off of the rails. Know the signal step values and determine their appropriateness. Adjust as necessary. Provide the calibration procedures to convert the transducer values into engineering units.

2.3.6 Describe the Signal Preconditioning

Most signals require at least some signal preconditioning. This can be high and/or low pass filters, signal amplifiers or attenuators, EMI protection, DC Offset adjustment, etc.

Any special circuitry necessary to precondition the signal should be included in this document.

2.3.7 Describe Diagnostics and Probe Points

Describe how you know that a transducer is working correctly. Where do you probe? What are the expected values? What diagnostics have to be passed?

Generally, there is a particular order that needs to be checked to assure that everything is functioning correctly. Document it.

2.4 Interface Control Document

So, you've ordered all of the parts and they are sitting there on the floor in boxes. How are you going to fit them all in? The job of the Interface Control Document is to describe all of the distribution, cabling and interfaces between all of the subsystems.

Mechanical drawings should be available to describe the physical layouts of the transducers and other components. All of these signals have to reach their controllers, even in a distributed control system. Now is the time to plan how they get there. It is sometimes easier for cabling to run from the subsystems to a central distribution point and route the signals from there. Multi-layer printed circuit cards can replace rats nests of wiring with a clean alternative.

2.4.1 Determine how the Subsystems Interact with the Endpoints

The wires must be correctly sized and long enough to allow assembly and disassembly of the subsystem components. They must not chafe on through holes which can cut through the wiring and cause shorts, etc. Once the electrical engineers have managed these details, you should be left with one or more cables that plug the subsystems together. The pin numbers and signals must be documented.

2.4.2 Determine how the Subsystems Interact with Themselves

There will undoubtedly be interactions from subsystems. As an example, the High Voltage Power Supply Controller requires a Vacuum Go/NoGo signal from the Vacuum Controller. This assert high signal should be routed across a copper line between the 2 systems, along with its return path.

2.4.3 Wiring Diagrams

Unambiguous wiring diagrams should be drawn up that allow you to trace a signal from the controller all the way to the transducer, knowing the cable that the signal goes through and the pin number that it comes out on. All channels will be verified for correct operation during assembly and that verification documented.

2.5 Closed Loop Control vs. Open Loop Control

Open loop control provides the commands necessary to control the system, but without feedback that the commands successfully completed. Closed loop control provides verification that the steps successfully completed. Open loop control is cheaper and simpler to implement. Closed loop control is more expensive and complicated, but safer and more certain.

As you examine the trade offs between these two types of control, factor in the cheaper cost of commissioning and repair of closed loop systems. Remember too the greater degree of confidence that the customer will experience from a closed loop system.

3 Control System Implementation

Ideally, the documentation listed above will be provided before implementation of the control system starts – NOT! It never happens that way, so plan on it. Putting a good

system together is more iterative than linear and there are things that the control system implementer can do to assist in the design process.

3.1 Simulation

If feasible, hardware simulation can ease the process of implementing the control system. The idea is to externally provide the analog and digital signals that will be monitored and controlled by the actual control system. This can be done by writing a software simulation of the physical process and interfacing with analog inputs, analog outputs, digital inputs and digital outputs that will provide the "transducer" signals to the control system.

Modeling the physical process provides a great deal of insight into how the process will actually operate – more so than will be gained by simply writing the control system. Timing considerations, fault modes and subtle interactions will find their way to the surface during this exercise.

Providing a hardware simulator will provide the ability to "freeze time" allowing the programmer to leisurely step through events that in reality take place in milliseconds. Proper operation of the control system can be validated in a way that would never be possible with actual hardware.

Having a physical simulator allows the coding of the control system to proceed before the physical hardware is actually available, thus collapsing the amount of time necessary to complete a system. As mentioned, problems are flushed out early. Sometimes these problems require a change to the physical hardware. Finding them early minimizes the effects on the hardware schedule.

When you have a hardware simulator, you can safely generate fault conditions to test the exception processing of your control system and to provide complete code coverage during the testing phase.

Since you are interfacing with actual analog and digital signals, you can complete the control system implementation using the actual controllers that will be used in the final system. This leaves you with the perfect system to train the operators as it is using the actual hardware and software that will be used on the real system but without the dangers inherent with training on the real system. It also gives you the ability to externally generate fault conditions as part of the operator training. The operator will be better able to recognize them and deal with them in the real world, because (s)he has already seen them and dealt with them on the simulator.

3.2 Minimal Model

The importance of minimizing and simplifying the model cannot be over emphasized:

- Increasingly complex systems are handled by increasingly simplifying abstractions. The simpler we make the components, the more complex the systems that we can successfully build.

- Simple models are easier to understand and implement. They have fewer mistakes and operate more reliably.
- Over 80% of the lifetime costs of a system goes to maintenance. Every line of code that can be removed from a system recovers 4 times the cost in maintenance.
- If we can't make it simple, chances are that we really don't understand it. This should be a huge red flag as we approach implementation.

The process of simplification is straight forward:

- Examine the system until you really understand it.
- If you have questions, get answers until no further questions exist.
- List the variables that affect a process and understand what they do.
- Role play the "If I were a High Voltage Power Supply (for example), what would I do?

Use the Occam's Razor approach. Once you have a working model, take out anything that won't break the model until you can't take anything else out. What is left is the minimal model.

Examine for opportunities to replace 2 steps with a single step. Keep an open mind. "It's always been done this way" is not a justification for doing it that way again. Understand the reasons for doing it that way.

When coding the system, first get it to work. Then, get it to work right. Then get it to work fast, cheap, reliably, etc.

Once you get your code to work, apply the Occam's Razor approach to the code. Comment out header files and recompile the code. If it recompiles with out them, you don't need them. If it generates error messages, look at the lines of code that generated the message to understand why the header file is required. Do the same for variables until you have a minimum model. Every line of code that is left in will have to be maintained for the life of the project. Long function calls with huge commented out sections obscure the underlying algorithm. Remove them. Don't leave in a single line that isn't required for the code to work properly.

When you have working code, go back and examine each and every function call and make sure you understand why it is there and how it works. Examine the error returns and understand why you need to process them. Don't be afraid to use gotos. This is industry, not academia. Efficient and correct algorithms are the deliverables, not bragging rights because you never use gotos. Examine your entry and exit conditions to assure that all possible inputs are correctly handled.

Only after you have code that works right do you attempt to optimize it. Your code is laid out in simple, readable fashion and the process of optimization is likely to obscure it. Be sure to keep a copy of the readable code before you start as you may have to scrap your changes and go all the way back to it. Whenever possible, opt for legibility over

efficiency. Remember that 80% of your time is spent in 20% of the code. If the optimization is not in that 20%, the savings aren't worth it. Leave it clear to understand. When you have obscured the functionality because of the optimization, document it. Even if you personally wrote the code, the next time you look at it you won't understand what you did unless you document it. If someone else optimized it and didn't document it, it's going to take a long time and careful analysis to understand.

Use Hungarian Notation for variable names, even in Linux systems. These names convey information that can help you avoid problems.

Choose a consistent parameter passing convention. If you use "row, column" everywhere but in one function where you use "column, row", you will inevitably invert these parameters when you call that function.

Use typedefs and STRICT type checking. Research done at Shell in the 80's proved that up to 90% of control system errors could be caught at compile time. It is better to find your errors during design than after the system is fielded.

Compile early and often. It is surprising how difficult it is to find two errors than it is to find just one. Know when you have introduced an error and correct it then and there. Run your code and make sure that fixing one thing hasn't broken something else.

Apply the principle of "Well Trusted Code". Spend the time to completely debug sections that you will use over and over. You will wind up with a Tinker Toy set that you can quickly plug together to create increasingly complex components.

3.3 Code Coverage

Figure out how to execute and validate EVERY line of code. Unvalidated exception processing can cause worse exceptions than the ones you are trying to handle. This is one of the areas that a hardware simulator can help with. A simulator can cause a power supply over-temperature without building a bonfire under the unit. Check your entry and exit conditions again because you usually validate your code using normal conditions. What happens in upset conditions?

3.4 Power On Self Test

Implement POST code early. The hardware has to be working correctly for the code to work correctly. On one project, I found 4 different hardware module failures with the POST code. I would have spent a great deal of time questioning the correctness of the software had I not known that the hardware had failed.

Implement version checking as part of the POST code. You have to be sure which version of the hardware and software are running together to know that the system is working correctly.

3.5 Diagnostics

Build in the diagnostics – don't tack it on as an afterthought. Work with the electrical engineers to determine what is needed to validate correct operation.

Modern exception processing can mask the fact that you have problems. Implement exception logging in your exception handlers. Time tag the error messages so that you can relate error events back to bobbles in the process.

4 Documentation

The documentation that you provide will distinguish an exceptional well thought out system from a working system that was simply cobbled together. If you ever have to fix a system, the documentation will make the difference between success and failure. If you ever want to replicate the system, the documentation will be the difference between quick and easy vs. practically starting over.

4.1 Accuracy and Completeness

Inaccurate and/or incomplete documentation can be worse than not having any documentation at all. If you are going to do it, do it right.

Accuracy is paramount. Inaccurate documentation can kill people and/or damage property. When someone goes to the documentation, they expect and deserve correct answers. Go over the documentation with a fine tooth comb and assure that it is accurate.

The documentation should provide complete and unambiguous information about a system.

You can use a check list to assure that all of the pieces have been included.

To assure that the documentation is unambiguous, it is best to have it validated by someone other than the original author. We see and hear with our minds, not our eyes and our ears. Someone who is intimately familiar with the process will make silent assumptions about the system that will bridge gaps in the documentation where someone who is unfamiliar with the process will realize that something has not been properly explained.

A good wordsmith should provide the final arrangement of the information. (S)he will have greater expertise at setting up tables of content and indices than the technician that originally did the documentation. The documentation is meant to be used, so make it possible to use it. Don't take the Microsoft approach. They document everything, but it is a major effort to find the correct documentation.

4.2 Dynamic Documentation

During the design and implementation phases, everyone needs the latest and greatest set of documentation. One of the problems with paper documentation is that it gives a static view of the state of the system at release. As documentation is distributed, it is

imperative that it be disseminated to all concerned parties simultaneously. Having even a single person working from a previous version can be disastrous to both schedule and budget.

Basing the documentation on a network accessible database, with dynamic output has the advantage of providing the most current view of the system available. It also allows you to take advantage of the integrity checking available from modern databases. Enforced integrity helps assure both the accuracy and completeness of the documentation.

4.3 Web Based vs. Paper

Disseminating the documentation via web pages makes the documentation available globally. We have seen in the past that many pages can be dynamically generated straight from the database. Hyperlinks can be automatically embedded to provide quick and consistent navigation to exactly the information that you are looking for.

Setting up a centralized web server allows you to give the documentation the widest possible accessibility. All of the engineers, designers and implementers have simultaneous access to the most recent and accurate information. Making the web accessible database the master repository of all project information makes it the fiducial source of information. It will contain the absolute standard for information about the current state of the project:

- The information is fiducial.
- It can be automatically partially verified for accuracy.
- It can be automatically verified for completeness.
- It can be shared and widely disseminated.
- It can be secured against inappropriate access.
- Automatic hyperlinks provide easy navigation.
- It can be linked to purchase orders, deliveries, bills of material, etc.
- All information is accessible from a single root location.
- The final website can reside with the system itself.