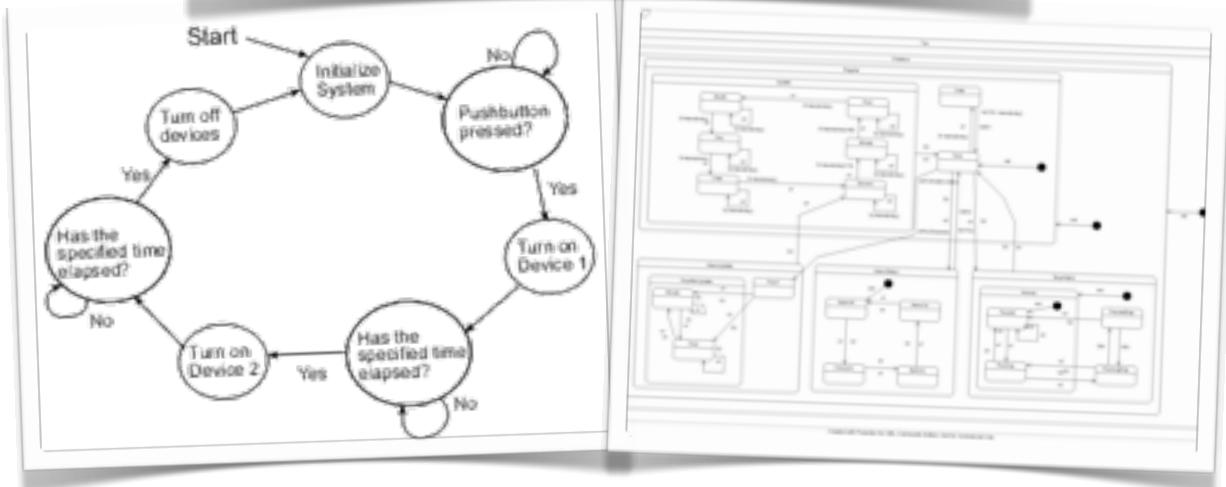
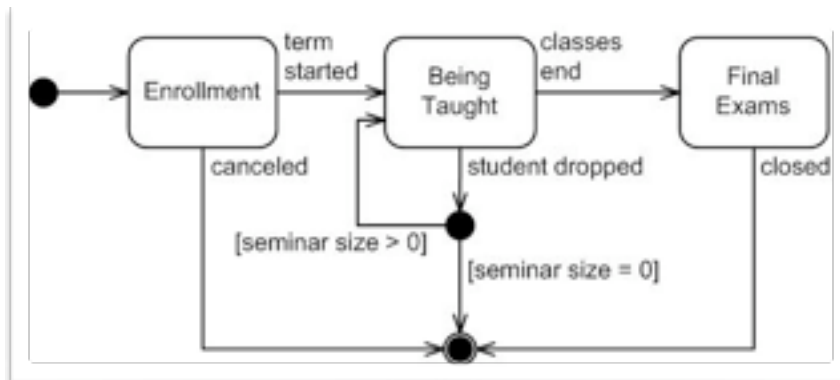


State Machine vs. PLC programming



State machines meet procedural
programming

ICD
30.03.09

A different approach

The automation studies for engineers have traditionally taught procedural programming. This is largely because of common practise and the limitations a PLC typically has, with little processing power and small memory size.

With modern controllers a typical task can be divided into smaller, concurrent tasks. Tasks can have different priorities. In addition, with a communication framework which supports events and alarms handling, such states can be handled concurrently.

A short table will show the differences between the capabilities:

	Controller	PLC
Processing speed	Practically unlimited	Usually limited, higher speed very expensive
Failure rate	IPC very sturdy	Sturdy
Parallell processing	State machines, most common solution	Practically impossible
Memory size	Practically unlimited	Limited
Adressing values in programs	Textual representation	Memory blocks and cells
Real-time capability	RTOS very common	Set run frequency, but dependent on program execution time

IPCs have traditionally been harder to program, with few integrated tools available for creating the different components necessary for a controller based control system.

Procedural programmings shortcomings

Typically, PLCs are programmed with circuit-diagram-like ladder logic, with pictograms showing checks, statuses and values. These programs are then executed top-down, with a number of checks and operations being carried out during the run (execution of the sequence).

If a check has been evaluated and afterwards something happens that changes the expected action, this will not be acted upon until the next

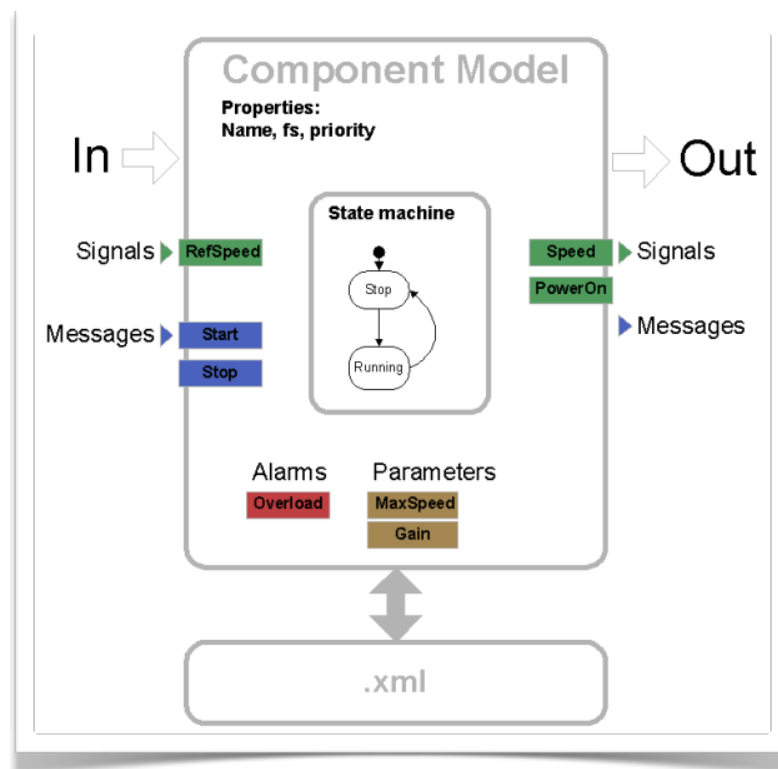
run is carried out. In a real-time system, this can have the result that the correct action is postponed unnecessarily. With state machines, the evaluation is done again and again at a set frequency and the correct action can be carried out immediately.

With a component based structure, an almost indefinite number of state machines can run concurrently and the system response thereby improves significantly.

Able platform and tools

With CDP from ICD AS, it is now very simple to create advanced and intricate control systems. The developer does not need to know C++ or any other programming language to make a simple control system.

(Over 80% of all programs on today's PLCs consist of 20 program steps or less.)



The state machine component has a state going in and another state going out. The check or operation inside the component is based on its signals (input) and it may produce signals or messages (output). Alarm handling may also be included.

The setup and configuration of parameters is held in external files (XML) and the configuration can thus be changed without changing the application itself.

Code intolerance no problem

Even if the CDP platform is C++ based and has ample possibilities for customisation and expansion by able programmers, the beginner or those with less complex system can use the code generator to make the whole application, including the .cpp, .h and .XML files. So if you don't understand what those file types are, don't worry, you don't have to! The code generator creates all the source code necessary to make your system work. You can of course re-use older components and only do the applicable changes in the setup files, making application development very quick.

Testing and simulation

The concurrent nature of a component based system also simplifies testing and simulation, and with the tools included with CDP values can be graphed and investigated on the fly without any system hardware in place. The control system behavior can also be tested by setting values directly into the components. This can be done on a system in HIL testing mode and the effects will be visible immediately.

This testing ability is valuable, as can be inferred here:

PID Loop Performance

Surprisingly, as many as 75% of control loops actually **increase variability!** Many control loops simply do not do their job. Setpoints are not followed, valves swing around, creating oscillations, and many loops are disabled by the operator: placed in MANUAL mode.

Poor performance happens in large part due to control loop configuration issues. Studies of control loops in the process industries give some insight into the root cause of these issues. For example:

- 30% of DCS Control Loops Improperly Configured
- 85% of Control Loops Have Sub-Optimal Tuning
- 15% of Control Valves are Improperly Sized

(ExperTune PID configuration errors, 081212)

With the correct tools this can be tested and found immediately when the system has the ability to graph, simulate and change system behavior on the fly.

To make sure your development process is as effective as possible, contact me at bh@icd.no or check our website www.icd.no