Example of using TEAM ICLP APIs for Discrete Device Control

Author: Sushil Birla

April 25, 1997

This example describes the application of a discrete control module for on-off control of simple bistate devices (Appendix B) that provide peripheral functions in a numerically controlled machine tool.

Consider a case where the NC machine has been retrofitted with third-party supplied work-handling automation, including two hydraulically actuated, two-position mechanisms, named, Loader and Unloader. These are modeled as bistate motion devices (Appendix B).

Let their sensing, actuation, and control be under a small discrete control module, named LUNL (Figure 1), whose sequence of operations was originally specified in some manner conforming to IEC 1131-3, and subsequently translated into a canonical control plan, named CPlunl (Figure 2).

Let the coordinated motion of the servo-controlled axes be under another module, named, AG, whose sequence of motions was originally specified through a RS 274-D part program and translated into a canonical control plan, named CPag.

Let the coordination of LUNL and TG be under an overall machine control level discrete control module, named MC, whose control logic, originally specified in some IEC 1131-3 form and translated into a canonical control plan, named CPmc.

LUNL has the work-performing objects, Loader and Unloader, and a controlling FSM, driven by CPlunl. Table 1 describes the normal sequence of operations, specified in CPlunl for a simplified hypothetical example case. It is not a part of the TEAM ICLP architecture. Events are messages received from MC or internally generated in LUNL. LUNL's FSM checks for an event and processes it, as specified through the system configuration parameters, typically at a specified repeating interval, consistent with the rate of change in the LLNL's application environment. The example assumes that operation of the NC machine, the loader and the unloader cannot be concurrent, and that the loader and unloader must be returned to their home positions before any function on the NC machine can be performed.

Thus, the TEAM ICLP architecture provides a unified, yet modular, framework for close interaction and integration of discrete control of the overall machine tool, continuous control of servo-motion, and discrete control of peripheral on-off devices.

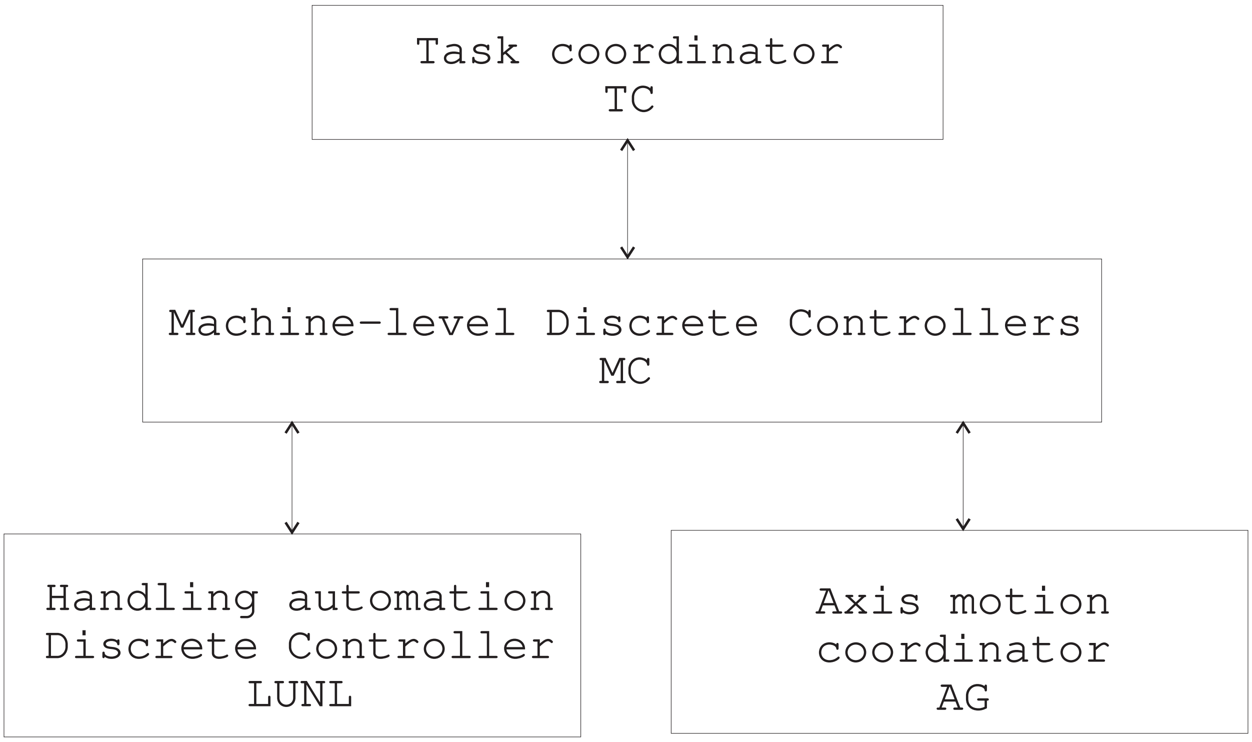

Figure 1: Overall information flow in the example machine tool controller

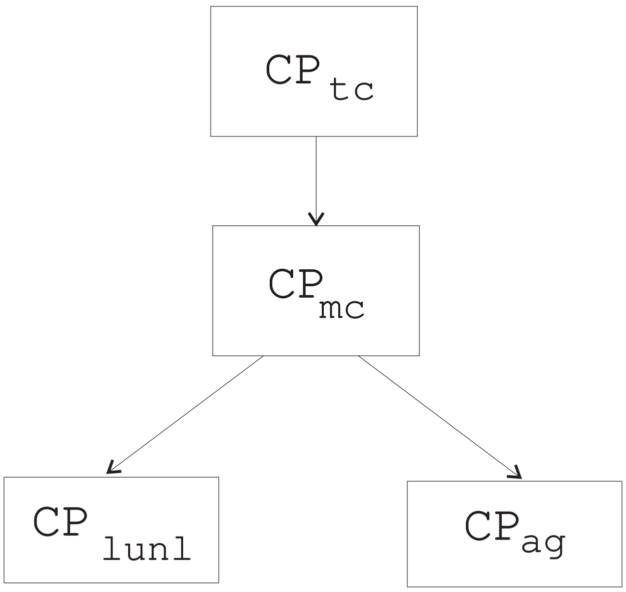

Figure 2: Hierarchy of FSMs and control plans in the example machine tool controller

Appendices

A Relationship of discrete control in TEAM ICLP with IEC 1131 model

A discrete control module in the TEAM ICLP architecture may be an O.S.-process, i.e., a process under the management of an operating system — known as *task* in IEC1131-3 [2, Figure 1; Section 2.7.2]. Alternatively, a discrete control module may be a lighter thread of execution, included in another O.S.-process. In either case, the TEAM discrete control module also satisfies the definition of a *logic control system* [1], where definite output signal states are functions of the input signals and the current state of the discrete control module. The TEAM discrete control module also satisfies the definition of a *sequential control system* [1], where the individual steps are processed in a predetermined order, progression (*transition*) from one sequence step to the next being dependent on a defined *condition* being satisfied. The TEAM discrete control module may be viewed as a subset of a *programmable controller* [1], i.e., one of many components in an automated system (industrial control system).

B Key classes of bistate devices

On-off devices may be sensors or actuators, subclassified on the delay between the state-switching event and the corresponding physical event, as follows:

1. Bistable sensor or actuator with no significant externally perceptible motion and negligible delay between the state-switching event and the corresponding physical event. Examples: *flood coolant valve*, *mist coolant valve* in a numerically controlled machine tool.

2. Bistable sensor or actuator with no significant motion but sufficient delay in action to require allowance in sequencing operations. Examples: *Tableclamp*, *ToolClamp* in a numerically controlled machine tool.

3. Bistable motion devices. Examples: Hydraulic or pneumatic cylinders with fluid flow controlled through a 4-way direction-control valve.

Functions for operation of bistable motion devices:
void advance();
void return();
set_motionState(STATE); //enum STATE: returned, advanced, is_returning, is_advancing.
STATE get_motionState();

Table 1: Simplified hypothetical state transition table for LUNL

CurrentState	Event	Action	NextState
LUNL returned	Advance unloader [1]	$obj_1- >$execute(…)	UNL advancing
	Advance loader [1]	$obj_2- >$execute(…)	L advancing
UNL advancing	Overtime [2]	$obj_3- >$execute(…)	Exception
	Unloader is advanced [2]	$obj_4- >$execute(…)	UNL advanced
UNL advanced	Return unloader [1]	$obj_5- >$execute(…)	UNL returning
UNL returning	Overtime [2]	$obj_6- >$execute(…)	Exception
	Unloader is returned	$obj_7- >$execute(…)	LUNL returned
L advancing	Overtime [2]	$obj_8- >$execute(…)	Exception
	Loader is advanced [2]	$obj_9- >$execute(…)	L advanced
L advanced	Return loader [1]	$obj_{10}- >$execute(…)	L returning
L returning	Overtime [2]	$obj_{11}- >$execute(…)	Exception
	Unloader is returned [2]	$obj_{12}- >$execute(…)	LUNL returned
Exception	Reset exception [1]	$obj_{13}- >$execute(…)	LUNL returned
…	…	$obj_i- >$execute(…)	…

Column explanations:
"LUNL returned" is the starting state, indicating LUNL is "home".
Event sources indications: [1] MC, [2] LUNL
Actions include functions of objects in the LLNL module, as specified in $obj_i- >$execute(…) $\forall i$, e.g.,
- get sensed inputs
- check conditions, e.g., "over time limit"
- set actuation outputs
- send LUNL state

Terminology used in the TEAM ICLP FSM model:
Transition: The quadruple (event, condition, action, next state);
where "condition" is omitted in this table for simplicity.
CurrentState and NextState are ProcessState objects.
StateRecord: A ProcessState object and its associated Transition objects.
STtable: Container class for a collection of StateRecord objects.

References

[1] *Programmable controllers – Part 1: General information*, National Electrical Manufacturers Association, December 1994. approved as an ANSI Standard, December, 1994.

[2] *Programmable controllers – Part 3: Programming languages*, National Electrical Manufacturers Association, December 1994. approved as an ANSI Standard, December, 1994.