PLCbatch - a Batch Process Control Tool on the PLC platform

Presented at the
World Batch Forum
North American Conference
Philadelphia, PA, USA
24 - 26 March 2008

Note: This is a reprint of the original paper submitted for presentation at the WBF.
Design of a Batch Process Control Tool on the PLC platform

Giovanni Godena
Applied Researcher
Jožef Stefan Institute
Jamova 39
Ljubljana, 1000
Slovenia
Phone: +386 1 477 36 19
Fax: +386 1 477 39 94
giovanni.godena@ijs.si

Igor Steiner
Process Control Project Manager
INEA d.o.o.
Stegne 11
Ljubljana, 1000
Slovenia
Phone: +386 1 513 81 17
Fax: +386 1 513 81 70
igor.steiner@inea.si

Janez Tancek
Project Engineer
INEA d.o.o.
Stegne 11
Ljubljana, 1000
Slovenia
Phone: +386 1 513 81 50
Fax: +386 1 513 81 70
janez.tancek@inea.si

Marko Svetina
S&M Manager
INEA d.o.o.
Stegne 11
Ljubljana, 1000
Slovenia
+386 1 513 81 07
+386 1 513 81 70
marko.svetina@inea.si

KEY WORDS
PLC Recipe execution, Tabular recipes, Phase state model

ABSTRACT

A new concept was developed of a tool for recipe-based control of batch processes according to the S88.01 standard, executed on the PLC platform, without PCs coupled within the real-time control loop. The concept features simultaneous execution of more recipes with automatic allocation of units, support of parallel as well as selective branches and the execution of the state-transition algorithm of individual phases, extended by the notion of superstates to improve the abstraction level. The main benefits of the concept are in the relocation of the execution of recipes from the inherently unreliable PC platform to the considerably more reliable PLC platform and in the simplification of recipe-based batch process control systems, without essentially reducing the expressive power and the abstraction level.
1. INTRODUCTION

Today there is a broad consensus among the manufacturers of automation equipment, service providers in this field and end users on the need and rationality of compliance with the S88.01 [1] standard in batch process control, regardless of the level of automation of this control. However there are also certain problems concerning the application of the S88.01 standard and Batch tools based on it, and these are:

1. Unsatisfactory reliability of the personal computer (PC) platform and real-time communication loop (can be bypassed through redundancy but incures significant extra cost).
2. Poor adjustment of the existing tools to the majority of small and medium-sized projects.
3. Unsatisfactory time behaviour of the PCs (concerning execution speed and time determinism).
4. Too low expressive power of phase behaviour model, which is shown in:
   - a frequent need to memorize the current step on state transition for returning to the same step, which is a non-transparent low-level concept, and per se a frequent source of errors,
   - complicated logic of individual states, which also often behave as state machines.

As a consequence of some of the abovementioned problems the contractors of batch process automation projects often decide for different solutions of executions of recipes on the programmable logic controller (PLC) platform, which are mostly over simplified and for this reason lose most of their expressive power, offered by S88.01 based tools.

The most frequent example is the execution of a recipe as a simple step sequence with a single Procedural Control Entity (PCE) in an individual step, containing all the functionality needed at that step. In this case the only theoretically possible flexibility is in changing the sequence of PCEs, which in most cases would result in an unfeasible sequence and therefore we cannot speak about flexibility of the solution anymore. This problem is solved in practice in two ways. The first way is in attaining flexible behaviour of ample PCEs through their parameterizing, and the second one is in the application of a recipe composed of small PCEs, that is phases, with still only a single phase per step appearing in a recipe, while other processing is performed in PCEs executed separately and concurrently with the recipe. The problem of the second solution lies in difficult attainment of synchronization between the main thread phases and concurrent PCEs. In practice this is mainly solved with hard-coded interdependence, which presents an obvious example of inadequate (too high) coupling and it is a great potential source of errors.

In the following of this paper we describe the design of a batch process control tool on the PLC platform, which is designed to avoid the abovementioned problems as much as possible.

The paper is structured in the following manner. Chapter Two gives the requirements and limitations that were to be considered in planning the approach. Chapter Three describes the concept of tabular presentation of SFC (Sequential Function Chart) recipes. Chapter Four states the entity behaviour model, and chapter Five presents a short summary.

2. REQUIREMENTS AND LIMITATIONS

Considering the starting points presented in the introduction and the limitations of the target implementation platform, we have defined the following requirements that the new batch process control tool should satisfy:

1. The tool should be executed on PLC and PC platforms, namely all real-time control functions, including recipes, should be executed on PLC platforms, and only non-real-time functions, such as for example Equipment Editor, Recipe Management, Batch Management, and Archiving, should be executed on PC platforms.
2. Recipes should be represented by means of SFC notation or in a sort of tabular notation. The tabular notation would probably be easier to be implemented on the PLC platform than the (graphic) SFC notation, which speaks in favour of using a table. Certain experiences from the past batch control projects with tools according to the S88.01 standard also made us think in this direction, when we encountered a problem in understanding or in the mastering of the SFC
notation by operators, and we explained the behaviour of SFC recipes to them by means of tables, and thus we started to think that the tabular notation might in some cases be even more acceptable for operators than the SFC notation.

3. Recipe hierarchy will have to be reduced due to platform limitations – we decided for two recipe levels: a procedure level, which contains unit procedures, and a unit procedure level, which contains phases. By leaving out the operation level we sacrificed some reuse and a part of the structure. We estimate that this is sufficient for the majority of real (small and medium-sized) projects where the new tool would be used.

4. The behaviour of procedural control entities should comply with the S88.01 standard, but with the introduction of certain model improvements (extensions). Based on many years of experience in modeling procedural control of continuous and batch processes we have reached the conclusion that the offered phase behaviour model is too simple, and the consequence is too complex content of individual elements of the model.

5. States and modes propagation mechanism of batch control tools is one of more complex aspects of these tools. Therefore, a simple but efficient state propagation mechanism among procedural control entities should be determined for this tool.

6. The tool should be designed in manner to achieve the highest possible level of portability (to various PLC platforms).

3. CONCEPT (NOTATION) OF A TABULAR RECIPE

The concept is based on mapping the SFC recipe design to a table and the execution of this table on the PLC by the table interpreter. The tabular record of recipes will be named Tabular Batch Recipes (TBR). The shortest possible description of the basic operational concept of TBR interpreter is: *the condition for recipe transition to a new row is that all dominant phases of the current row that are not repeated in the next row as the same instance, are terminated.*

The abovementioned phase parameters need to be explained – dominance and instance, before describing the SFC to table mapping rules, and the table execution algorithm.

3.1 System Phase Parameters

Two system phase parameters are needed for the execution of tabular recipes or step transition of these recipes.

The first parameter defines phase dominance, therefore it defines whether the termination of a phase represents a condition for recipe transition to a new row. As a rule it also defines whether the phase has its own (intended) completion (for example dosing a material) or it would be inherently executed infinitely (for example mixing). If the phase is dominant, the recipe row transition is possible only after termination of the phase; if the phase is dependent, the recipe row transition appears independently of this phase; if the phase is active at that time, the recipe interpreter stops it.

The dominance attribute value of an instance of an inherently dominant phase can be changed to dependent behaviour (i.e. we can take off the phase the decision on its completion and pass it to the recipe interpreter), but the opposite does not hold – an inherently dependent phase can under no circumstances obtain dominant behaviour, since it is not programmed to behave in that way.

The procedure itself for setting the value of the dominance attribute for phase instances is the following: recipe editor during the creation of a recipe copies the phase own (default) value of the dominance attribute from the phase control block (PCB) to dominance attribute for an instance in the recipe. If the default value of phase dominance (in a PCB) is "FALSE", the value of dominance for phase instances cannot be changed in the recipe (therefore the default value is also the only possible value); however, if the default value of dominance in PCB is "TRUE", the value of the dominance attribute for phase instances in the recipe can be changed during its editing.

The second parameter, named Instance, defines the instance of an individual phase in a recipe. This parameter is needed when the same phase appears in several consecutive recipe table rows. In this
case the phase behaviour on row transition depends on whether the instance of this phase in a new row is the same as its instance in the current row (which has the same meaning as the appearance of this phase in a long thread in the SFC), or there is another instance of this phase (which has the same meaning as the repetition of the same phase in several consecutive steps of the SFC).

3.2 Mapping of a SFC Recipe to a Table

Figure 1 shows an example of a recipe, and Table 1 shows its mapping to a table. In this example the dominant phases are marked with an asterisk (*). Algorithm for mapping the SFC into the table is the following: we move along the SFC thread containing most phases. Each of these phases is mapped into a distinct table row, and also all phases that appear concurrently in parallel SFC threads are mapped into the same row. Therefore, the mapping of our example is the following:

1. Phase A has no concurrent phase and therefore it is mapped alone into the first table row.
2. Phase B has a concurrent phase G, therefore these two phases are mapped into two cells of the second table row.
3. Phase C has two concurrent phases D and G, therefore these three phases appear in three cells of the third table row.
4. Phase E also has two concurrent phases D and G, therefore these three phases appear in three cells of table row four.
5. Phase F has a concurrent phase G, therefore these two phases are mapped into two cells of table row five.
6. Phase H has no concurrent phase and therefore it is mapped alone into table row six.
7. Phases I and J are concurrent and are mapped into two cells of table row seven.
8. Phase I appears alone (without any concurrent phases) and therefore it is mapped alone into table row eight.

3.3 Behaviour of the Table Interpreter

The execution of the recipe table is performed rather intuitively. For the considered example it goes as follows:

1. In the first row there is only a dominant phase A, therefore the termination of this phase implies the recipe row transition.
2. In the second row there are two dominant phases B and G, but only the termination of phase B implies the row transition because the phase G is repeated in the next row as the same instance. On row transition the interpreter does not influence the execution or the state of the phase G.
3. In the third row two dominant phases C and G appear as well as a dependent phase D. The row transition depends only on the termination of the phase C as the phase D is a depended phase, and the phase G is repeated in the next row as the same instance. The state of the phase G is not
affected by row transition, and the same applies to the phase D as also this phase is repeated in
the next row as the same instance.
4. In the fourth row two dominant phases E and G appear as well as a dependent phase D. The row
transition depends only on the termination of the phase E as the phase D is depended, and the
phase G is repeated in the next row as the same instance. The state of the phase G is not
affected by recipe row transition, whereas the phase D is stopped by the interpreter (as it is not
repeated in the next row).
5. In row five two dominant phases F and G appear. The row transition occurs on termination of
both phases as none of them is repeated in the next row as the same instance (in this case they
are not repeated at all).
6. In row six there is only a dominant phase H, therefore its termination implies the row transition.
7. In row seven two dominant phases I and J appear. The row transition occurs when both phases
are terminated as none of them is repeated in the next row as the same instance.
8. In row eight there is only a dominant phase I, therefore its termination implies the row
transition.

3.4 Possible Structures of Recipes

The structure of a recipe is a two-level structure – the procedure owns its recipe, i.e. its higher-level
table, which contains unit procedures, and each unit procedure owns its recipe, i.e. its lower-level
table, which contains phases of this unit. The only limitation of recipe structure at both levels is that
determinism of conditions must be provided for the transition to a new step, which means that, at
any point of the recipe, only one of the currently active concurrent threads contains more than one
step (and a step can be a phase or a new concurrent branching). The abovementioned limitation has
no consequences in most of real cases, but in the remaining cases consequences are shown in the
form of a partly reduced concurrency in the recipe regarding the highest theoretically possible
concurrency; in other words, the execution of individual recipe parts can be slowed down.

3.5 Specific Transition Conditions and OR Recipe Branching

It is inherent for the presented tabular notation of the recipe that it supports "AND" branching and
transition conditions based on the termination of given (dominant) phases. In the recipe system we
certainly also expect the option of "OR" recipe branching and specific conditions for recipe
transition, such as for example reaching a defined value of a given process variable. These and
similar requirements can be satisfied with the introduction of special phases. Here we will introduce
two of such special phases: phase Condition and phase Branch.

3.5.1 Special Phase Condition

Special phase Condition serves for the implementation of situations, when a logical expression,
performing a relational operation on a given process variable, appears as a condition for step
transition in the SFC recipe. In TBR this is solved by special phase Condition, which repeatedly
performs the relational operation, and when the comparison result becomes "TRUE", the phase
completes. If this phase was the only dominant phase in the row where it was placed, we attained
the desired recipe behaviour – the step transition will occur when the condition is satisfied, i.e. at
"TRUE" value of the relational operation.
In the Condition phase the comparison type (relational operator) and comparison objects (what is
compared with what) need to be stated. We certainly want a general phase, i.e. we do not want to
introduce a distinct Condition phase for each process variable that we want to introduce into a
recipe for comparison. This is solved by the corresponding phase parameterizing with the following
parameters:
• First operand type [variable | constant ]
• Id/value of the first operand (considering whether it is a variable or a constant)
• Relational operator \[ = \mid <\mid <\mid >\mid <= \mid >= \]
• Second operand type \[ variable \mid constant \]
• Id/value of the second operand (considering whether it is a variable or a constant)

3.5.2 Special Phase Branch

Special Phase Branch serves for the implementation of "OR" recipe branching, when the execution of the recipe is threaded in two possible directions considering the given condition (the value of the logical expression, as in the Condition phase). Such behaviour of the Branch phase is attained by enabling the phase write access to the system variable containing the value of the recipe step. The phase decides based on comparison and therefore it has the same five parameters as the Condition phase, and additional two:
• Next step for comparison result "TRUE",
• Next step for comparison result "FALSE".

Using the Branch phase, correct structure is not provided by recipe editor as with the SFC graphic editor, but the recipe creator must pay attention to it (as is the case with programming languages lacking structured programming constructs)

We can also implement the Jump special phase, which performs an unconditional change of a step in the execution of the recipe. This phase has only one parameter, Next step.

4. ENTITY BEHAVIOUR MODEL

PCE states arise from the behaviour model as it is proposed by the S88.01 standard. The model is extended in certain elements for attaining better properties, especially concerning simplicity, understandability and reliability of the application software.

4.1 Phase Behaviour

In order to attain as simple phase logic building blocks as possible, it is essential to build a model where phase building blocks will be merely simple sequences as far as possible, without any internal states or threading, and without the need to memorize the entry point to the sequence (for example whether we came to Running.ENTRY sequence from state Idle or from state Restarting as actions in these two cases differ). In compliance with this and arising from the behaviour model of procedural control entities in our domain-specific modeling language ProcGraph [3] we defined the phase behaviour model, which is an extended state machine model and is shown in Figure 2. The extension comprises the following properties:
1. Introduction of new elementary states Evaluating Starting Conditions (EvaluatingSC) and Evaluating Restarting Conditions (EvaluatingRC), to achieve distinction between holding causes, phase initial conditions and phase restarting conditions
2. Introduction of nested states (elementary states and a number of superstates).
3. Introduction of a very fine granularity of the processing (action) structure:
   a. State sequences of actions, five different possible sequences for each state:
      • ENTRY actions, which are executed only once on entry to a given state (in Figure 2 the states having ENTRY actions are marked with an E),
      • LOOP actions, which are executed cyclically all the time the phase is in a given state (in Figure 2 the states having LOOP actions are marked with a L),
      • EXIT actions, which are executed only once at the exit from a given state (in Figure 2 the states having EXIT actions are marked with a X),
      • ALWAYS actions, which are executed during the state activity and during transitions entering this state (in Figure 2 the states having ALWAYS actions are marked with an A),
      • TRANSIENT actions, which are executed only once at transient states (in Figure 2 the states having TRANSIENT actions are marked with a T);
b. Transition actions, which are basically specific ENTRY actions of the target state, therefore they are a part of this state, but they are executed before the ENTRY actions of this state (in Figure 2 the transitions having actions are marked with an A).

Actions of all sequences have duration, i.e. they are not instantaneous. This applies not only to ALWAYS actions or LOOP actions, as it is the case in the Statechart notation [2], but also to ENTRY, EXIT and TRANSIENT state actions and transition actions. The advantage of this is that there is no problem of synchronizing individual activities.

ENTRY and LOOP actions are not executed if at the entry to a state the condition for termination is satisfied or if any failure cause is present, but EXIT actions are executed in any case.

4.1.1 State and Transition Types

We know different state types, which are divided according to two criteria. According to processing criteria states are divided in the following way:
- Quiescent states are states without any processing (left part of Figure 2),
- Active states are states that contain certain processing (right part of Figure 2).

According to duration criteria states are divided in the following way:
- Transient states are those states that contain only one sequence, and when it is executed, a transition to another state occurs. Each active transient state contains one TRANSIENT sequence.
- Durative states are those states in which a phase normally remains for a longer time. The processing of active durative states is divided into several sequences which can be of ENTRY, LOOP, EXIT and/or ALWAYS type.

![Figure 2: Phase state transition diagram](image-url)
Quiescent states (which are all also durative states) are *Idle*, *Complete*, *Stopped*, *Aborted* and *Terminated*.

Active durative states are *Running*, *Held*, *Executing*, *Operating*, and active transient states are *EvaluatingSC*, *Holding*, *EvaluatingRC*, *Stopping* and *Aborting*.

Transitions are divided in active and inactive transitions. Active transitions contain certain processing. These transitions are: from *EvaluatingSC* to *Running*, from *Starting to Held*, from *EvaluatingRC* to *Running*, from *EvaluatingRC to Held*, and from *Held to EvaluatingSC*. Each active transition contains one sequence of transition actions.

### 4.1.2 Complex Transitions

Complex transitions are series of state and transition sequences on the way between two durative states, or between a durative state and a branching transient state, which is a switch on the way to one of the two possible target durative states. In other words, complex transitions are integrated processing wholes (composed of a number of sequences) during the execution of a phase state machine. The execution of a phase state machine is performed by the Phase Logic Interface (PLI), which checks the need for activating one of the complex transitions that are available for each active durative state. When the conditions (or causes) for the occurrence of a complex transition are met, the PLI executes a complex transition by consecutively activating all sequences of this complex transition.

### 4.1.3 Phase Logic Sequences

Phase logic sequences present a framework for the modularization of batch control software. Considering the phase behaviour model, shown in Figure 2, the theoretical number of sequences for each phase is 58, namely four sequences per each durative state, and there are 9 durative states (elementary state or superstate), one sequence per each transient state, and there are 5 transient states, and one sequence per each transition, and there are 17 transitions. It certainly turns out that many sequences are not needed and thus their number can be strongly reduced.

The first, trivial, level of sequence reduction refers to all sequences of quiescent states and transitions between two quiescent states. All these sequences are not needed in the model and they were excluded. The second level of sequence reduction is executed in the following way: all the sequences that appear together in a complex transition, and at the same time each of them appears only in this transition, can be replaced with a single sequence. Considering the abovementioned reductions of behaviour model, the phase contains 22 sequences: 17 in states and 5 in transitions.

### 4.1.4 Further Reduction of Phase Behaviour Model

The described phase behaviour model with 22 sequences presents the maximum possible range of elements or sequences, therefore the minimum possible granularity of the execution of phases. In case a certain programmer (organization) thinks that the abovementioned 22 sequences are too many as he/she has built a simpler phase behaviour model (with lower number of sequences), he/she should only configure the sequences that he/she does not want to use to be inactive. The PLI ignores (does not activate) sequences configured in this manner. Theoretically the programmer can also program applications by using only one sequence, although in most cases this would look more like hacking than programming. He/she can also use *exactly the same five active sequences* as the currently available batch process control tools use according to the S88.01 standard.

### 4.2 Recipe Behaviour Model

The same as phases, a recipe is also executed as a state machine. The essential difference is only that at a recipe state *Running* is composed of two substates—*ExecutingStep* and *AdvancingStep*. In the state of *ExecutingStep* the interpreter first resets (puts in state *Idle*) all phases of the current step, then it activates them and waits that all conditions for the transition to a new row in the recipe table
are met. When this happens the interpreter passes to the AdvancingStep state, where the current step is "cleared" (all dependent phases that are not repeated as the same instance in the next step are stopped by the interpreter's command).

During particular recipe states the interpreter influences the phases of the current step by sending them commands. For example in the recipe state ExecutingStep the interpreter sends commands Reset and Start to phases, during recipe state AdvancingStep it sends command Stop, during recipe state Holding it sends command Hold, during recipe state Stopping it sends command Stop, and during recipe state Aborting it sends command Abort.

Figure 3 shows recipe state transition diagram.

**Figure 3: Recipe state transition diagram**

### 4.3 State Propagation Mechanism among Procedural Control Entities

Batch process control tools according to the S88.01 standard that are executed on PCs have integrated state and mode propagation mechanisms of procedural control entities. A similar but simpler holding propagation mechanism is integrated also in the new tool concept.

In higher-level recipes, which are composed of unit procedures, recipe holding (due to a command) is propagated downwards to all unit procedures; however, holding of individual unit procedures is not propagated upwards to the recipe.

In lower-level recipes there is a propagation mechanism, which is executed through two (system) phase parameters:

- Parameter that defines whether holding of this phase is propagated upwards to the recipe containing the phase.
• Parameter that defines whether holding of the recipe containing the phase is propagated downwards to this phase.

The proper combinations of these two parameter values in individual phases can provide various behaviours: when a defined phase is held, the recipe goes to holding or not, and when the recipe is held, certain phases go to holding, while other remain running.

5. SUMMARY

A new concept was developed for recipe-based control of batch processes according to the S88.01 standard.

Usually recipes are created on a batch server – normally industrial PC – which acts as a batch execution engine. Within a real-time control loop batch server communicates with a PLC during recipe execution. Phase logic, and basic control of the process are performed by the PLC. Since PCs are inherently unreliable, redundancy of execution engine and communication lines is used to increase safety and reliability of the operation.

PLCs are one of the most reliable equipment components within production process. They are becoming more and more capable. Based on these facts the new concept for batch execution was developed.

Within the new approach the creation of recipes and batches is done on a PC and then downloaded to a PLC. PLC itself becomes batch execution engine while still taking care of phase logic, and basic control of the process. The need for real-time communication loops and the need for an industrial PC is eliminated. The need for redundancy is drastically reduced. The new concept is simple, cost effective, reliable and understandable, yet sufficiently powerful for most cases of batch process control.

The concept is based on the mapping of the SFC recipes to the tabular notation. It features simultaneous execution of more recipes with automatic allocation of units, support of parallel as well as selective branches and the execution of the state-transition algorithm of individual phases. The phase behaviour model is extended by the notion of superstates to improve the abstraction level. As a consequence of the extended behaviour model the phase logic is composed of really simple sequences, without branches based on sequence execution history, or elements of state-transition algorithm built into application logic, or remembering the step counter in the Running state before holding, in order to continue the execution at the appropriate point after restarting.

An additional benefit of the introduced state-transition model is that the programmer is allowed to select a particular subset of the predefined states and superstates, in order to define his/her own model of phase behaviour.

REFERENCES