– For Unlimited Distribution –

P4 Style Examples

Curtis Flippin
Flippin Engineering

November 6, 2009

1 Engineering Talking Paper

This is the same example that was included in the P4 Technical Description. It is an Engineer's SWAG[1] at a chemical process monitoring system. Engineers often must become instant experts in a subject. This P4 paper is typical of the talking papers that one might use to meet with area experts and get a better feel for the detailed requirements and design issues.

```
(* P4 Simple Process Control Monitor Example
   Curtis G. Flippin
   22 October 2009
   P4 Pseudocode (c)2001-2009 *)
(*
   System physical plant consists of one or more production units.
   A production unit accepts two reagents that are mixed in a
   specific proportion in a reactor vessel under a specified
   production standard temperature and pressure, PSTP.
   The reagents react to form a new chemical that is output
   from the reactor vessel.
   The process is continuous.
*)
   define:  system   as    g(lambda) chemical production unit;

(* System Identification
   Significant a priori knowledge of the system has been modeled
   and incorporated via a set of nominal and optimum operating
   parameters for each of the following production factors. *)
   define:  r1_r2    as    input reagents proportions in GPM;
   define:  c1_r3    as    production output quality characteristic one;
   define:  c2_r3    as    production output quality characteristic two;
   define:  f_rate   as    production output flow rate in GPM;
```

[1] Scientific Wild Ass Guess

```
   define:  pstp     as    production standard temperature and pressure;
   define:  a_spd    as    vessel mix agitator speed;

(* System Dynamics
   The chemical reaction process combines two reagents, r1 and r2,
   in unequal proportion, r1_r2, to produce a product, r3, with
   the desired qualities, c1_r3 and c2_r3, in a quantity defined
   as the flow rate, f_rate. The process is non-linear.
   Production process controls consist of the following reactor
   vessel controls.
*)
   define:  ctl_temperature   as    reactor temperator control;
   define:  ctl_pressure      as    reactor pressure control;
   define:  ctl_speed         as    reactor agitator speed;

(* System Performance
   The objectives of the Process Control Monitor are to aid
   human operators in controlling the reaction process and to
   log system performance in a manner that will allow
   consolidation with reaction control process logs for
   later use as a learning tool for an adaptive control
   system model. Performance is tracked as an index that is
   a function of all the variable parameters plus the
   control deltas over a defined ideal sampling rate.
*)
   define:  ip    as    performance index which is a function of
                        (r1_r2,c1_r3,c2_r3,f_rate,pstp,a_spd,
                        delta[temperature,pressure,speed]);
   define:  d_ip  as    delta(ip) memory;

   function:   delta_ip(c1,c2,f,t,p,d_ip);
   (* Calculate delta(ip) from production factor arguments
      and return in d_ip.
      input arguments
         c1    quality characteristic 1
         c2    quality characteristic 2
         f     product flow rate
         t     reactor temperature
         p     reactor pressure
      output arguments
         d_ip  delta of latest ip to new calculated ip.
         d_ip is retained data that is used by this function
         and elsewhere to determine the modification direction
         information that is sent to the operator.
   *)
   end:  delta_ip;

(* System Analysis
   Operators perform the process analysis in response to data
   they receive from the process control monitor. The monitor
```

```
   does not determine corrective actions. It tracks performance, ip,
   and compares the ip to both optimal and nominal performance
   standards. Operator notices are transmitted to the operator
   indicating the current level of system performance. There are
   three levels of notices, optimal ip, nominal ip, and sub-nominal
   ip. Notices include all system parameters and are logged along
   with a timestamp.
*)
   define:  ip_optimum     as    optimum performance notice;
   define:  ip_nominal     as    nominal performance notice;
   define:  ip_subnominal  as    sub-nominal performance notice;

   procedure:  delay;
      (* This procedure is a stand-in for the sampling rate
         control process. An ideal rate will help to create
         parameter delta's that are well above ambient noise.
      *)
   end:  delay;

   procedure:  monitor;

      do:   load initial known production parameters;

      (* Monitor runs until the shutdown process orders
         monitoring to stop.
      *)
      until:   stop-order received from shutdown;
         do:   read sensors r1_r2, c1_r3, c2_r3, f_rate, a_spd,
               temperature, pressure;
         (* Mathematical function f(ip) is not yet defined *)
         do:   calculate ip performance index;
         do:   calculate delta_ip(,,,,,d_ip);
         if:   ip is outside optimum performance limits;
               if:   ip is outside nominal performance limits;
                  do:   test ctl_temperature, ctl_pressure,
                        ctl_speed for sub-nominal readings;
                     if:   any are sub-nominal;
                        ?? we may want to track time between
                           sub-nominal alarms in order to
                           elevate the alarm level for
                           persistent performance problems.
                        ??
                        do:   set alarm status for sub-nominal
                              controls;
                     endif:
                  do:   send ip_subnominal performance notice;
               else:
                  do:   send ip_nominal performance notice;
               endif:
         else:
```

```
                do:   send ip_optimum performance notice;
          endif:
          call: delay to maintain an ideal 1 cps sample rate;
       repeat:

    end:  monitor;

end:  Process Control Monitor
```

2 System Test Plan

An informal system test plan proposal is put forth using P4. Note that it contains only comments.

```
(* System Test Planning
   Curtis G. Flippin
   Senior System Engineer
   Flippin Engineering
   Paylo Sellhi, California
*)
(*
   System Test Planning Proposal (Draft)
   3 November 2009
*)

(* The Project Design and Development Phase has just begun
   and attention must be given to the System Test Plan.
   Engineering Testing will begin very soon and continue
   through Qualification Testing. We will need to have
   a testing framework in place to define and manage the
   tests. This document provides a proposed starting
   point for Test Planning.
*)

(* The Level 1 Test Plan should contain the following
   elements:
   1. Test Objectives
   2. Participating Agencies and Responsibilities
   3. Test Group Organization and Functions
   4. Operating Procedures, Methods, and Controls
   5. Test Schedules
*)

(* The Detailed Test Plan will need to address:
   1.    Detailed Test Objectives
   2.    Test Responsibilities
   3.    Test Support
   4.    Items and Steps in Test
   5.    Test Methods
```

```
   6.    Typical Test Measurement
   7.    Test Results Data Reduction
   8.    Criteria for Success
   9.    Test Operations Plan
   10.   Test Operations Schedule
   11.   Test Operations PERT Charts as Required
*)

(* TEST OBJECTIVES
   To include:
   1.1   Functional Requirements
   1.2   Hardware Capabilities and Limitations
   1.3   Reliability
   1.4   System Technical Documentation
   1.5   Evaluation of Operations and Maintenance Plans
   1.6   Safety
   1.7   Training
*)
```

3 Merge Sort Algorithm

P4 is a good tool for defining non-mathematical algorithms. It is generally more useful because it eliminates the dependency on a specific programming language implementation.

```
(* Two-Way Merge Sort Algorithm
   Curtis Flippin
   Flippin Engineering
   4 November 2009
*)
(*
   The two-way merge sort basically divides a list
   having at least two items into two roughly equal
   sublists. Each sublist is sorted by recursively
   applying the merge() function which will produce
   a number of sublists. Eventually, all the sublists
   become merged into a single sorted list.
   This is an (nlogn) sort and its been around for
   many years.
*)
   function:   merge(list);
   (* Two-Way Merge Function *)

      if:   the list contains less than two items;
         do:   return (list) because it is sorted;
      endif:

      define:  left     as    a list;
      define:  right    as    a list;
```

```
      define:  result   as    a list;

      (* The list is partitioned into two sublists *)
      define:  middle   as    integer (list_length / 2);
      do:   put all list items up to middle into left;
      do:   put all list items after middle into right;

      (* Divide and Conquer by recursively performing
         merges on successive sublists *)
      do:   left = merge(left);
      do:   right = merge(right);

      (* On the last pass, merge the two remaining lists
         into one final sorted list *)
      if:   the last item in left > the first item in right;
            do:   return (result = merge(left) + merge(right));
      else:
            do:   return (result = left + right);
      endif:

   end:  merge();
```

4 Code Style

If need be, P4 documents can be made to look very much like code. This short temperature scale conversion program is an example with a code-like style.

```
(* Celsius/Fahrenheit Temperature Converter
   Curtis Flippin
   Flippin Engineering
   5 October 2009
*)

(*
   This program accepts a temperature as input and
   types the conversion in both directions.
   Celcius to Fahrenheit and
   Fahrenheit to Celsius.
   The user never needs to specify which temperature
   scale is intended.
*)
   function:   getinput(number);
      do:   accept an input number and
            return number;
   end:  getinput();

   procedure:  temperature_converter;
      (* Accept input temperature to convert *)
      getinput(temperature)
```

```

      define:  celsius  as  number;
      define:  fahrenheit  as  number;
      (* Convert Fahrenheit to Celsius *)
      do:   celsius = (5 * (temperature - 32)) / 9 ;
      (* Convert Celsius to Fahrenheit *)
      do:   fahrenheit = (( 9 * temperature) / 5) + 32 ;

      (* Print the results *)
      do:   print temperature, " to Celsius = ", celcius;
      do:   print temperature, " to Fahrenheit = ", fahrenheit;

   end:  temperature_converter;
```

5 A Question of Logic

This last example shows a slightly different style. The more common Symbolic Logic symbols are defined via P4 functions. There is no real program. Functions are used as a vehicle for describing the logic definitions of these symbols.

```
(* A Question of Logic
   Curtis Flippin
   Flippin Engineering
   6 November 2009
*)
(*
   Symbolic Logic defined as functions.
   The arguments are hypotheses that may
   also be complex symbolic logic terms.
   The functions return a conclusion of
   true or false based upon evaluation of
   the argument(s).
*)(*
   (A ^ B) is ^(A,B)
   (A v B) is v(A,B)
   ~A is ~(A)
   A -> B is ->(A,B)
   A <-> B is <->(A,B)
*)(*
   (A ^ ~B) becomes C = ~B, ^(A,C)
   ...and so on.
*)(*
   Altogether, a simple but not really
   useful exercise except to illustrate
   using P4 functions to describe a
   discrete, limited linear grammar.
*)
   function:   ^(a,b);
    (* Logical connective 'AND', (A ^ B) *)
```

```
   if:   a = true;
      if:   b = true;
            do:   return true;
      else:
            do:   return false;
      endif:
   else:
         do:   return false;
   endif:
end:  ^();

function:   v(a,b);
(* Logical connective ’OR’, (A v B) *)
   if:   a = false;
      if:   b = false;
            do:   return false;
      else:
            do:   return true;
      endif:
   else:
         do:   return true;
   endif:
end:  v();

function:   ~(a);
(* Logical connective ’NOT’, ~(A) *)
   if:   a = false;
         do:   return true;
   else:
         do:   return false;
   endif:
end:  ~();

function:   ->(a,b);
(* Logical implication ’IF...THEN’, A -> B *)
   if:   a = true;
         do:   return true;
         (* Means B is also true *)
   else:
         do:   return false;
         (* Means nothing can be implied about B *)
   endif:
end:  ->();

function:   <->(a,b);
(* Logical connective ’IF...AND ONLY IF’, (A <-> B) *)
   if:   b = true;
         (* Then A is true *)
         do:   return true;
   else:
```

```
            (* Otherwise, A is false *)
            do:   return false;
        endif:
    end:  <->();

end:
```