

A Structured Interactive Approach to Embedded Control

Johan Eker and Anders Blomdell

Department of Automatic Control
Lund Institute of Technology
Box 118 S-221 00 Lund
{johane|andersb}@control.lth.se

Abstract

Real-time control systems implemented with traditional methods are usually both hard to maintain and reuse. This paper presents a real-time software development environment for implementation of control system. A new dedicated language for writing control applications is presented. The language is designed to support features characteristic to control implementation and reuse of code. An interactive run-time system which allows on-line editing on running systems is also presented.

1. Introduction

The traditional way of implementing real-time systems using languages such as C or C++ gives deficient support for reuse of code. This means that embedded control software often has to be constructed from scratch, even for minor changes of system requirements to existing implementations. In the field of automatic control most implementations have a similar internal structure. To take advantage of this fact is one the key ideas behind PÅLSJÖ, a software environment for development of real-time systems.

To support implementation of features that are characteristic to control systems a dedicated language PAL (Pålsjö Algorithm Language) has been developed. Control algorithm can in most cases be described either as periodic tasks or as sequences, PAL supports both types of algorithms. Furthermore, the language supports data-types such as polynomials and matrices, which are extensively used in control theory.

PÅLSJÖ has been designed to support rapid prototyping of control systems. The engineer off-line defines a set of blocks which later at run-time can be instantiated and connected to form a control system. Blocks in a running system can be replaced without having to stop the system.

The system consists of two main parts; a compiler and a run-time system. The run-time systems provides a text interface for the user and a network interface for data transmission. PÅLSJÖ is primarily designed to run in a host-target configuration. Stand alone tools for on-line plotting of control signals have been developed. Reuse of algorithms is possible through the block library facility.

2. Related Work

There are several systems which all aims in the direction of greater support for implementing real-time software. There are among others the Onika/Chimera [10] system from Carnegie Mellon University, the ORCCAD [9] environment from INRIA in France, ControlShell [8] from Real-Time Innovations, Inc. and SattLine [7] from Alfa-Laval Automation. These are all quite extensive systems with graphical user interfaces and a lot of features.

In the development of the PÅLSJÖ environment the main goal has been to provide a simple but yet powerful method to implement control algorithms. The language PAL is be used to describe control blocks. No C or C++-code is needed to glue the system together. We have put more emphasis on a convenient textual description of control algorithms. We therefore believe that our approach can be used also in other control engineering systems.

3. PAL

3.1 Background

During the construction of the PÅLSJÖ system, it was soon apparent that C/C++ lacked many features needed to express control algorithms in a clear and natural manner. Therefore a new language was designed to make the semantic gap between control theory and implementation of algorithms as small as possible. PAL is not intended to be a full fledged

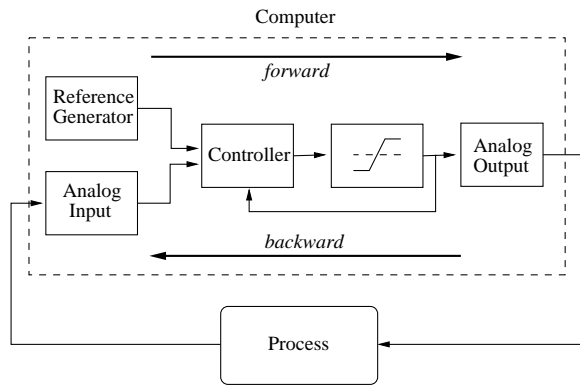


Figure 1 A simple computer controlled system.

language for everyday programming.

Control algorithms in PAL is described in terms of blocks, that read input signals and calculate output signals. Furthermore a block has parameters that can be set by the user and internal states.

To find out how blocks should be executed in a run-time environment we looked at some typical control structures as the one in Figure 1. This system consists of a five blocks that are connected together. The analog-input block is used to convert a measured analog signal from the process to a digital signal. The next block is a reference value generator, which supplies the controller with the desired process output. The output signals from those two blocks are connected to the control block. The control block calculates a control signal which propagates to the an actuator model. The actuator model simulates the behavior of the actual actuator and in the case where the control signal exceeds the limits of the actuator it will cut the control signal. The modified control signal is then fed to the analog output blocks, which writes to the actuator device.

The calculation of the complete system can be divided into two parts. First all inputs are read and a new output signal is calculated and written, and then the states of the system are updated. In PAL those two parts are referred to as the forward sweep and the backward sweep. The blocks are said to be swept in a forward direction to calculate an output signal and then in a backward direction to update the states of each block. In the backward sweep the limited control signal is fed back into the controller, which then can take appropriate action. In the case where the controller is of PI-type, this corresponds to adjusting the integral part to avoid wind-up.

Since there is a logic order of execution of the blocks It is possible for the run-time system to schedule the blocks and thus avoid synchronization problems that arises when each block runs as a separate process.

3.2 An Overview of the Language

The constructs used in PAL have many different roots; the module concept comes from Modula-2, the way to return function results is borrowed from Eiffel, the way to express Grafcet charts comes from the IEC-1131 standard [6] and the overall look has its roots in Algol. Modules are the top-level structuring concept in PAL. Modules are used to package blocks that are somehow related to each other and inside blocks, procedures and functions are declared. Identifiers declared in one module are not visible in other modules unless they are explicitly imported.

Blocks correspond to the black boxes that control engineers uses as their main abstraction view. The main characteristics of a block is that it has inputs and outputs. It may also have parameters that reflects the parameters used in algorithms (e.g. the gain K), dimensions, and derived parameters.

The scalar data-types are boolean, integer, real, dimension and sampling interval. Dimension is a new data-type which is used for dynamical allocation of aggregate data-types. Sampling interval is the current sampling time. There are also three aggregate types; array, matrix, and polynomial. To define variables in a block the following syntax is used.

```
<name> : <interface modifier> <datatype>;
```

The interface modifier states how the variable <name> shall interact with the environment; it can be either input, output, or parameter. If no interface modifier is given the variable is regarded as an internal state.

3.3 Two Examples

Implementation of a PI-controller with anti-windup is the first example. The syntax and basic module and block structure are shown. The second example is an implementation of a simple sequential algorithm.

PI-Controller The execution cycle for the PI-controller can be divide into two parts. One that reads the input signals and calculates a new output signal and another part that updates the internal states of the controller. The pseudo code for this would look something like this:

```
loop
  /* Forward sweep */
  read input signals
  calculate output signals
  write output signal
  /* Backward sweep */
  update internal state
end loop
```

When writing this in PAL only the calculate output section and the update state section needs to be included. All writing and reading of input and output signals is automatically taken care of by the runtime system. In this PI-controller there are two input signal y and y_r , one output signal u and two states e and I . There are also two parameters K and T_i . Below is an example of how this PI-controller coded in PAL. The block is basically built in three sections; first all signals and parameters are declared, then the output signal is calculated and last the state is updated.

```

module Module1;
  block PI
     $y, yref, v$  : input real;
     $u$  : output real;
     $K, T_i, Tr$  : parameter real;
     $P, I := 0.0, e$  : real;
     $h$  : sampling interval;

    forward begin
       $e := yref - y$ ;
       $P := K * e$ ;
       $u := P + I$ ;
    end forward;

    backward begin
       $I := I + h * K / T_i * e + h / Tr * (v - u)$ ;
    end backward;

  end PI;
end Module1.

```

The parameters K and T_i and the sampling interval h are given values at run-time. It is possible to give default values to both parameters and signals as done in the example above for the state I .

Grafcet A function chart in GRAFCET for a boiler process is shown in figure 2. The process is tank which initially is empty. The sequence starts when the button B is pressed. A valve V_1 is then opened and water start flowing into the tank. When the water level reaches the height L_0 , the heating of the water starts. This will continue until the tanked is filled and the temperature is T . The next step is to open the valve V_2 and empty the tank of the hot water. This sequence can be expressed in PAL as below.

```

module Grafcet1;
  block Boiler
     $B, L_0, L_1$  : input boolean;
     $T$  : input real;
     $Tref$  : parameter real;

    initial step Init; end Init;
    step Fill1;

```

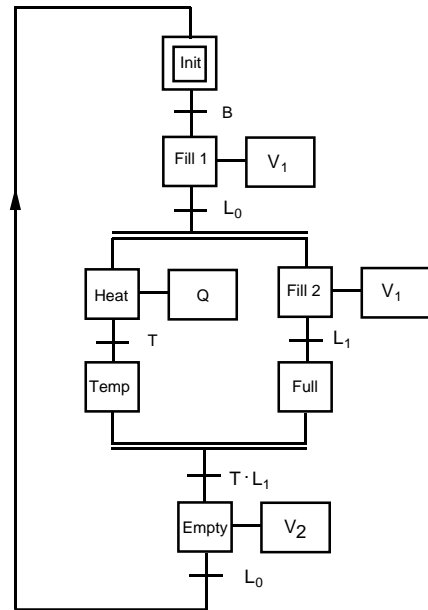


Figure 2 GRAFCET of a boiler process.

```

  store activate V1;
end Fill1;

step Heat;
  activate Q;
end Heat;

step Temp; end Temp;

step Fill2; end Fill2;

step Full;
  reset V1;
end Full;

step Empty;
  activate V2;
end Empty;

action V1; begin end V1;
action Q; begin end Q;
action V2; begin end V2;

transition from Init to Fill1 when B;
transition from Fill1 to Heat, Fill2 when L0;
transition from Heat to Temp
  when T >= Tref;
transition from Fill2 to Full when L1;
transition from Temp, Full to Empty
  when true;

transition from Empty to Init when not L0;

end Boiler;
end Grafcet1.

```

It is possible to have sequential and period algorithms combined in the same block.

4. Pålsjö Run-Time System

Compiled blocks are executed in the PÅLSJÖ run-time environment. Here instances of blocks can be created and connected together. To configure and manage blocks in Pålsjö the PÅLSJÖ Command Language, PCL is used.

The facility to export data to other programs via the network is supported in PÅLSJÖ. Today there are two tools developed for this purpose; A dedicated plot tool written in Erlang [1] and a client script in MATLAB, that can either plot data or simply log data for further analysis.

4.1 PCL

PCL is a command language used to configure the system on-line. The language has commands for allocating, deleting and connecting blocks. Modules written in PAL can be imported into the workspace and after that all block-types in the module are visible to the user, who now can instantiate the necessary blocks and connect them to form a complete control system. There exist a set of pre-defined blocks and the most important of those is the Sequence-block, which handles the execution of ordinary user-defined blocks. When a block is allocated in PCL by the user it must be set up so it belongs to a Sequence-block before it will be executed.

Below is a example session with PÅLSJÖ, where a system similar to the one in Figure 1 is created. First using the command use the necessary block types are made available. Next all blocks are allocated using the new-operator. All blocks are allocated as children to the sequence block, which manages the execution. To connect the output signal from one block to the input signal of another block the connect-operator (->) is used. Finally, the parameters of the blocks are assigned.

```
pcl> use Module1
pcl> use StandardBlocks
pcl>{
pcl*> process = new Sequence
pcl*> process.adin = new AnalogIn
pcl*> process.refgen = new RefGen
pcl*> process.control = new PI
pcl*> process.daout = new AnalogOut
pcl*>
pcl*> process.adin.out -> process.control.y
pcl*> process.refgen.out -> process.control.yr
pcl*> process.control.u -> process.daout.in
pcl*>
pcl*> process.tsamp = 10
pcl*> process.control.K = 2
pcl*> process.control.Ti = 0.5
pcl*> }
```

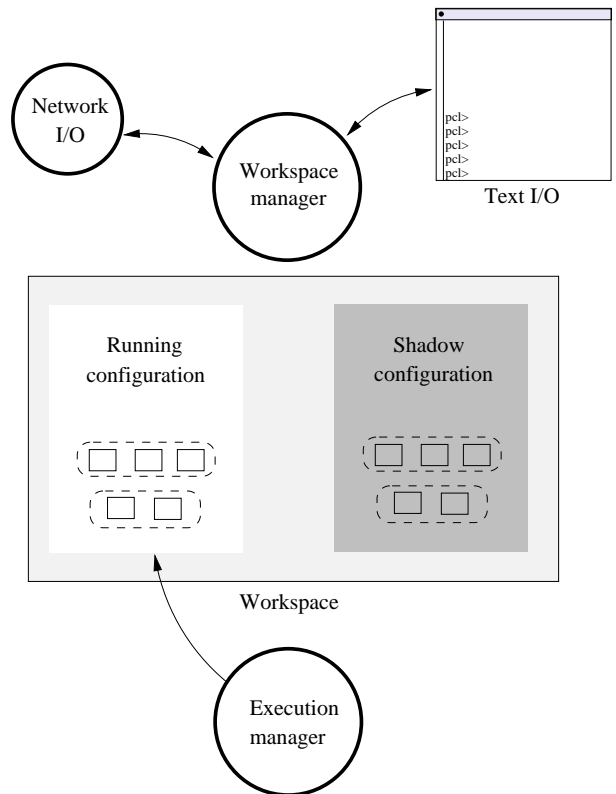


Figure 3 A schematic view of the run-time system, with one executing configuration and one back-up configuration.

PCL is also used for handling the export of data to the network. It is also possible to write PCL-macro that have zero or more input parameters.

4.2 The Shell

The run-time system support on-line editing in a running configuration. It is possible to replace blocks in a running controller configuration without having to stop the system. To be able to do this in a safe way the system always keeps two copies of each block. There is always one set of blocks that is running, called the running configuration, and another set called the shadow configuration, see Figure 3. The shadow configuration is a an exact copy of the running configuration. All edits to the system is made on the shadow configuration, which after the editing session is concluded, becomes the running configuration. If the changes results in a system that is not valid, then the edited shadow configuration is not allowed to replace the running configuration. Before the execution start all blocks are sorted with respect to the internal data flow. Algebraic loops are detected and reported.

4.3 Efficiency

A variable in PAL is defined with an interface modifier, and this information is used by the run-time system to optimize the performance. The execution

order of the blocks is determined by the data flow analysis and a schedule for the blocks is calculated. This means that the problem with the handling of shared data (i.e. input and output signals), is eliminated.

Since the blocks relative each other will run in a fixed order it is possible to avoid unnecessary copying and buffering of data.

5. Summary

In this paper an approach for implementing real-time control systems, using a new dedicated language PAL, is described. PAL supports many features that are necessary when implementing control real-time systems. A run-time system PÅLSJÖ, where PAL blocks can be executed is also presented.

Today PÅLSJÖ runs on Sun/Solaris, VME-m68k, and Windows NT. The compiler is currently only available for Solaris. Future plans include the ability to down-load new control blocks definitions at run-time, and development of an extended network protocol for distributed configurations. We intend to make the system, including source code and our kernel for VME-162 available via anonymous ftp in a future.

6. References

- [1] J. ARMSTRONG, R. WIRDING, and M. WILLIAMS. *Concurrent Programming in Erlang*. Prentice Hall, 1993.
- [2] K. J. ÅSTRÖM and B. WITTENMARK. *Adaptive Control*. Addison-Wesley, Reading, Massachusetts, second edition, 1995.
- [3] A. BURNS and A. WELLINGS. *Real-time Systems and their programming languages*. Addison-Wesley, 1990.
- [4] M. ELLIS and B. STROUSTRUP. *The Annotated C++ Reference Manual*. Addison-Wesley, 1990.
- [5] K. GUSTAVSSON. "An architecture for autonomous control." Technical Report, Department of Automatic Control, 1994.
- [6] INTERNATIONAL ELECTROTECHNICAL COMMISSION. *IEC 1131-3, Programmable controllers, Part 3: Programming languages*, 1992.
- [7] G. JOHANNESSON. *Object-oriented Process Automation with SattLine*. Chartwell Bratt Ltd, 1994. ISBN 0-86238-259-5.
- [8] S. A. SCHNEIDER, V. CHEN, and G. PARDO-CASTELLOTE. "The controlshell component-based real-time programming system." In *IEEE Conference on Robotics and Automation*, Real-Time Innovations, Inc., 954 Aster, Sunnyvale, California 94086, 1995.
- [9] D. SIMON, B. ESPIAU, E. CASTILLO, and K. KAPellos. "Computer-aided design of generic robot controller handling reactivity and real-time and real-time control issues." INRIA, Sophia-Antipolis, France, 1992.
- [10] D. B. STEWART, R. A. VOLPE, and P. K. KHOSLA. "Design of dynamically reconfigurable real-time software using port-based objects." Advanced Manipulators Laboratory, The Robotics Institute, and Department of Electrical and Computer Engineering, Carnegie Mellon University, 1993.