Specification of Industrial Digital Controllers with Object-Oriented Petri Nets

Ricardo Jorge Machado João Miguel Fernandes Alberto José Proença
Department of Informatics, Engineering School, University of Minho
4709 Braga codex, Portugal
rmac,miguel,aproenca@di.uminho.pt

Abstract – **The main purpose of this paper is to present an Object-Oriented PN model (shobi-PN) to specify industrial digital controllers. The shobi-PN model (Synchronous, Hierarchical, Object-Oriented and Interpreted Petri Net), was developed to support the use of hierarchy to model both the control unit and the plant of the systems. A CAD environment, SOFHIA, was developed to model digital controllers, to validate their properties and to simulate their behaviour. SOFHIA has an open architecture, which eases the integration of multiple code generator blocks to allow the implementation of the system in a wide range of technologies (hardwired, microprogrammed, programmed). Modelling of an Industrial Reactor control system is considered as a case study to ilustrate the model's applicability and capabilities.**

I. INTRODUCTION

The design of complex digital control systems is conceptually divided in two parts: the control unit and the plant. The behaviour of the control unit is usually described with an FSM, and the plant resources are directly controlled by the actions from the controller. To specify a digital control system, both the control unit and the plant should be considered by the specification model and the CAD environment. The complexity of the design task grows when the controller behaviour presents parallel activities.

There are several languages that allow the implementation of industrial controllers with Programmable Logic Controllers (PLCs), such as "*ladder diagrams*", logic languages, and GRAFCET [1]. Some of these languages are based on graphical formalisms, allowing the modelling of sequential or parallel control systems [2]. FSMs are used mainly for sequential controllers, while GRAFCET, based on PNs, is used for parallel controllers [3].

Some manufacturers make available tools to directly implement FSMs. FSM became so popular for specifying sequential controllers, due to the existence of specification tools and the generalized use of PLDs.

It is also possible to specify a parallel controller using FSM techniques: serially-linked controllers can be obtained by identifying sub-routines in the specification, while concurrently-linked controllers should be connected with semaphore bits or common lines [4]. These approaches are usually hard to apply, and can result in inefficient implementations due to the prepartitioning, which limits the concurrency to the number of FSMs used. Is is also hard to check for parallel synchronization problems (deadlock and multiple-sourcing).

Among the existing modelling paradigms, the PN-based one is the only that allows an easy specification of cooperative subsystems. PNs are a graphical language, easy to understand and a system modeled with a PN may benefict from a mathematical theory to formally verify its properties [5]. Using GRAFCET is similar to PNs, although slight differences make awkward to apply analysis techniques to the GRAFCET models [2].

The PN-based formalism presents the following characteristics [2]:

- Expressiveness, allowing the model to be built in a modular way;
- Explicit and clear modelling of activities and events;
- Graphical representation which can be used as an interface between users and designers;
- Strong theory for analysis and validation;
- Computer tools to aid the designers in some design tasks;
- Easy implementation of the system in a wide range of technologies.

An extension to SIPNs (Synchronous and Interpreted PNs) [6] was developed, supporting hierarchy on the PN models and the use of objects to model the plant. A full digital control system can be specified and tested, following a structured and incremental approach. SOFHIA (Software for Hierarchical Architectures), a CAD environment that covers all the design phases, was developed to directly support the shobi-PN model.

This paper is structured as follows. The characteristics of the shobi-PN model relevant for the purpose of this paper are presented in Section II.. In Section III., the SOFHIA CAD environment is introduced. A detailed example is considered in Section IV., which clarifies the presented concepts and shows the PN model capabilities to specify parallel digital controllers for industrial applications.

II. shobi-PN: THE PETRI NET MODEL

The writing of this paper has assumed that the reader is familiar with the PNs theory concepts. For details on PNs, please refer to [7].

A new PN model, shobi-PN (Synchronous, Hierarchical, Object-Oriented and Interpreted Petri Net), was developed to support the use of hierarchy and to model the control unit and the plant in the specification of digital systems.

The shobi-PN model presents the same characteristics as the SIPN (Synchronous and Interpreted PN) model [6], in what concerns synchronysm and interpretation, and adds functionalities by supporting object-oriented modelling approaches and new hierarchical mechanisms, in both the control unit and the plant.

This model embodies concepts present in Synchronous PNs, Hierarchical PNs, Coloured PNs, and Object-Oriented PNs [8, 9, 10].

In the shobi-PN model, the tokens represent objects that model plant resources. The instance variables represent the information of the plant and the methods are the interface between the control unit and the plant. The tokens may be considered as coloured, if SIPN tokens are viewed as uncoloured.

A node (a transition or a place) invokes the tokens' methods, when the tokens arrive at that node. Nevertheless, only the methods that have a direct relation with the hardware control signals are directly invoked in the PN. There are additional methods available at the objects' interface that are not used by the PN. These methods are invoked by the simulation software to visualize the contents of any structure of the plant in any state of the PN.

Each arc has one or more colours which associate the arc to the type of objects that are allowed to pass through it. This means that, for each plant structure, there is a well-defined path on the PN. This requirement simplifies the PN and limits the capacity of some places, since it is not needed that objects, that are not invoked, unnecessarily traverse the PN.

Hierarchy can be introduced in the models in two different ways:

- The control unit is modelled by the PN structure, and to implement the hierarchy on the controller, macronodes (representing sub-PNs) may be used;
- The plant resources are represented by the internal structure of the tokens, and the hierarchy can be obtained by *aggregation* (composition) of several objects inside one single token (a macrotoken) or by using the *inheritance* of methods and data structures.

Whenever several methods that use the same data structures are concurrently invoked to a given token in different nodes, it is necessary to support a replica mechanism. This mechanism allows a token to be replicated as many times as needed, so that it is structurally possible to concurrently invoke methods to the same token, but in distinct areas of the PN. This mechanism can be used as an elegant solution for a complex problem (the multiple-sourcing) that could be alternatively, but innefficiently, solved at the algorithmic level, by changing the PN structure. In Section IV., the use of this mechanism is used in the example.

A glossary for the shobi-PN model follows:

- Control Net: set of contiguous nodes and arcs of the shobi-PN that structurally corresponds to the SIPN without reinitilizations.
- Control Track: path defined by a token in the Control Net.
- Control Nodes: nodes (places or transitions) of the Control Net.
- Control Arcs: arcs of the Control Net.
- Closing Track: path defined by a token outside the Control Net.
- Closing Nodes: nodes of a Closing Track.
- Closing Arcs: arcs of a Closing Track.
- Closing Cycle: path defined by the movement of a token in the shobi-PN. It is composed by a Control Track and also, if applicable, by a Closing Track. It can be identified by the tracking of the colour associated with all the arcs of the cycle.
- Associated Net: SIPN structurally equivalent to the Control Net after the introduction of the reinitilizations for the uncoloured tokens.

III. THE SOFHIA CAD ENVIRONMENT

The SOFHIA CAD environment [11] is appropriate for specifying industrial controller systems with the shobi-PN model. At the moment, it feeds any ECAD package that accepts VHDL as input. The hierarchical PN specification is directly and efficiently mapped to boolean equations. This approach simplifies the VHDL code debbuging, since there is a direct correspondence between the original PN and the produced VHDL code. The complete framework is illustrated in Fig. 1.

All the tasks needed for industrial digital controllers design using shobi-PN-based specifications are completely supported by the SOFHIA environment. Among those tasks are: (1) formal verification of the properties of the model; (2) simulation; and (3) code generation to implement the system.

The SOFHIA environment is organized in 3 blocks: the Main Unit (MU), the Control Unit Manager (CUM), and the Data Path Manager (DPM) [1]. The MU is responsible for the full integration of the environment and for the interface with the user. The CUM is responsible for formally verifying the properties of the model and its correctness. It is also the CUM that generates code for the control unit implementation. The DPM generates a file describing the plant resources.

The most important inner blocks (in Fig. 1.), for the goals of this paper, are next described.

The Model Verifier checks if the input specification fulfils the rules imposed by the shobi-PN model. The tested rules include the definition of the initialization nodes, and the verification of the conservative property of the shobi-PN.

The Simulator module simulates (step-by-step or batch) the behaviour of the PN. The user can check the tokens' contents which aids to verify the correctness of the output values. For large or complex PN specifications, the formal verification may demand too many computer resources, which makes simulation one of the possible solutions. Another problem with formal verification is that it does not ensure that the system behaves as the user expected.

The Hw/Sw Partitioner selects the proper code generator block to generate descriptions of the system parts (control unit and plant) in intermediate languages. These descriptions will feed the CUMs and the DPMs to allow the parts to be synthesized in software and/or in hardware. This block allows the use of the SOFHIA environment for codesign. The algorithms for Hw/Sw partitioning are still under study.

The Codes Generator aims the generation of intermediate descriptions to feed the CUMs and the DPMs blocks. The ConPar Generator [12] is already implemented and it creates a file with the textual description of the specified PN in the intermediate ConPar language. This description is only related to the control unit of the shobi-PN and it will be used by the CUM. The DataPar Generator creates a file with the textual description of the specified PN in an intermediate language. This description is only related to the plant of the initial PN and it will feed the DPM (module not yet implemented). This CAD environment has an open architecture, which eases the integration of multiple code generator blocks to allow the implementation of the system in a wide range of technologies (hardwired, microprogrammed, programmed).

Several CUMs may exist in the environment, depending on the number of final representations for implementing the control unit. This allows several possible

[1] Since the environment is not only used for industrial control systems, the term "data path" was chosen to mean the "controlled part" of any specified system.

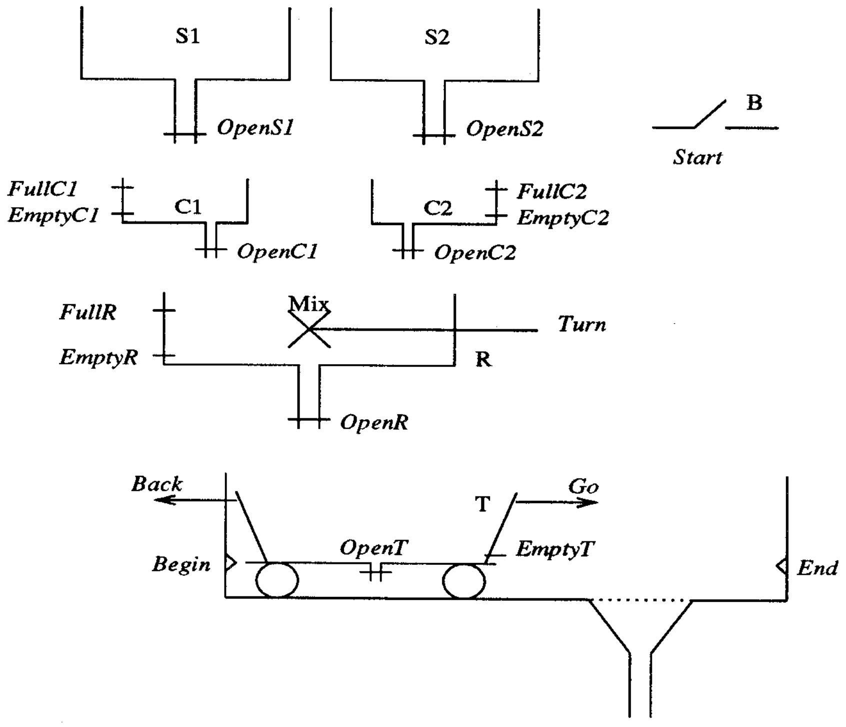

Fig. 2. The mixing and transport system for the reactor.

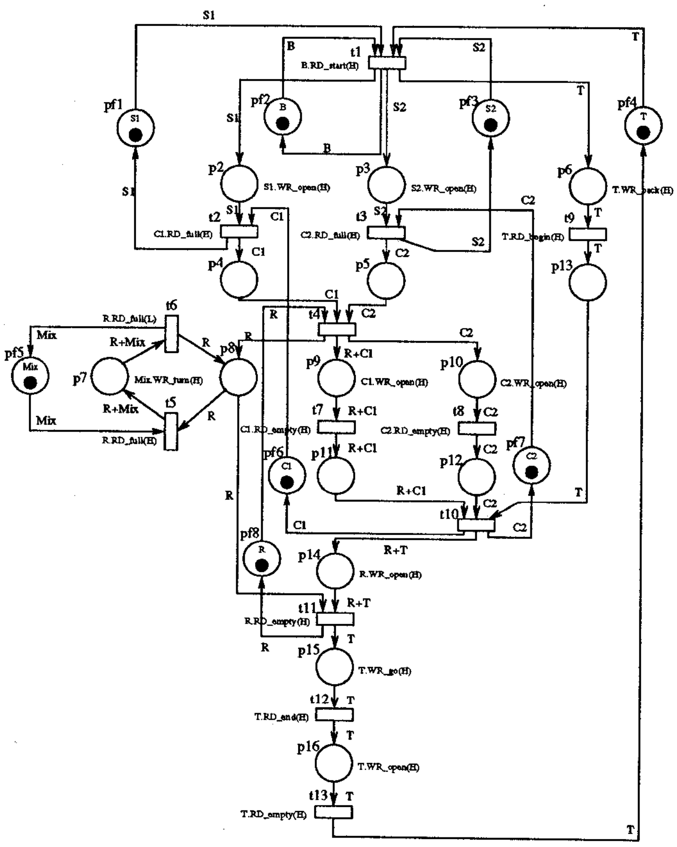

Fig. 3. shobi-PN for the industrial reactor.

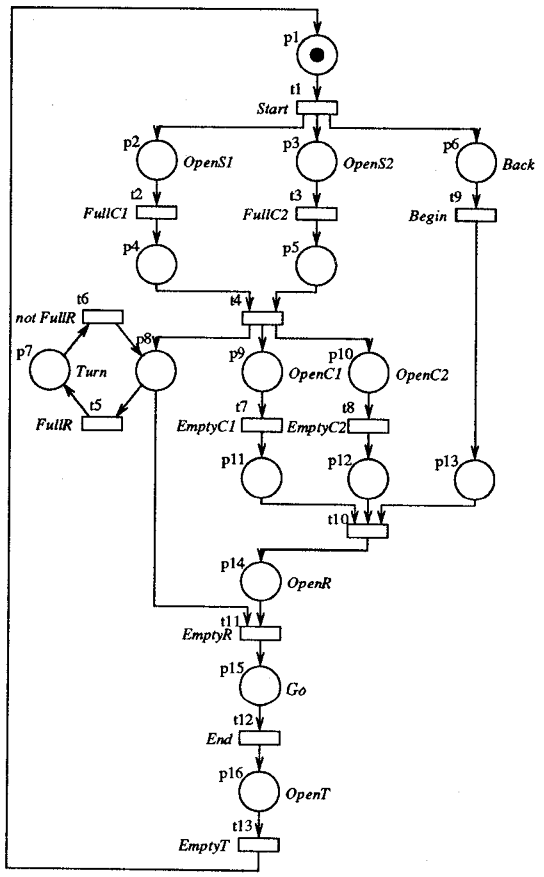

Fig. 4. SIPN for specifying the industrial reactor.

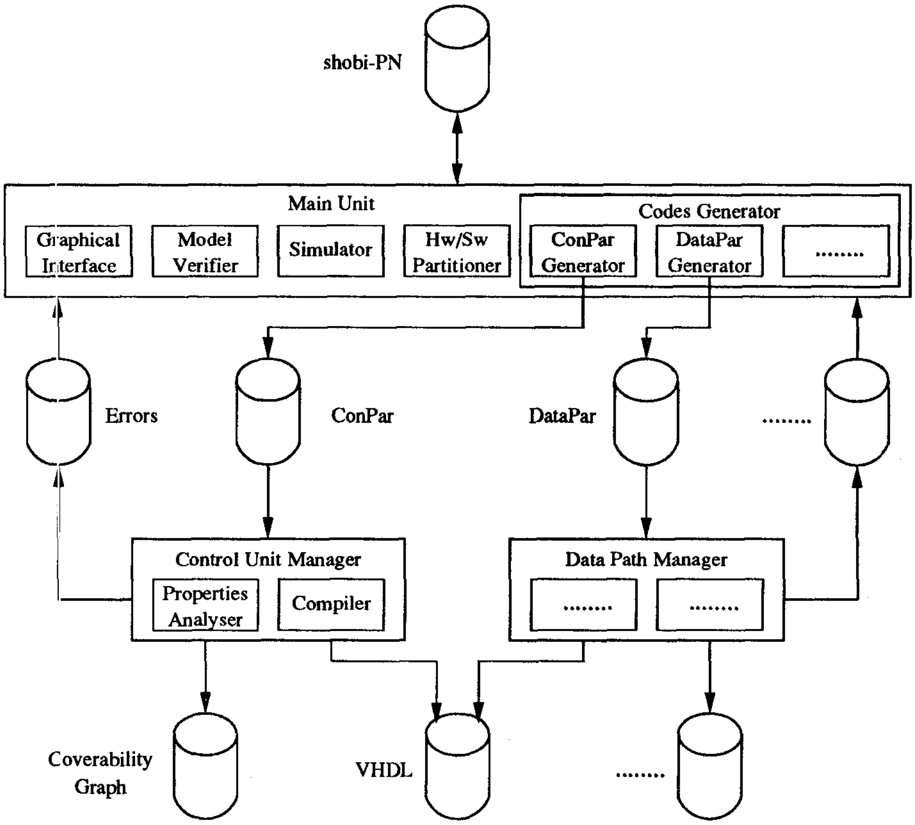

Fig. 1. The SOFHIA CAD environment.

implementations to be obtained (PLCs languages). The first already developed CUM accepts as input the specification of a control unit using SIPNs [6], written in ConPar, an intermediate language. For experimental purposes, the Compiler generates VHDL code, which allows the controller to be simulated or synthesized.

The Properties Analyser verifies if the input specifications are live and conflict-free, issuing a message to the user, whenever a problem occurs (deadlock or conflict). Since it is not appropriate to mark a place if it is already marked, that situation is also detected, clearly located and an adequate message is sent to the user interface.

Similarly, several DPMs may exist in SOFHIA, depending on the number of final representations to implement the data path. It may exist a DPM responsible for creating a VHDL file for the data path from an intermediate description supplied by the DataPar Generator.

IV. THE INDUSTRIAL REACTOR EXAMPLE

To clarify the concepts introduced in the previous sections, a detailed example is presented: the industrial reactor, which controls the behaviour of a system for mixing and transporting products. For further details about the system, please refer to [13, 14].

Before starting to build the shobi-PN, the industrial reactor's plant resources need to be modelled, by identifying the objects and defining its variables and methods.

After a careful analysis of the system in Fig. 2., eight different objects are identified to model the plant: two simple tanks S1 and S2, two tanks with sensors C1 and C2, one central reactor R, one transport car T, one industrial mixer Mix, and one button B. According to this selection, the corresponding classes to the identified objects are declared and coded (Fig. 5.).

The control unit is then specified with a shobi-PN, using instances of the previously defined classes. Each instance is invoked by methods existing in its interface during its travelling along the shobi-PN. This example uses two instances of the *tank* class (S1 and S2), three instances of the *tank_sensor* class (C1, C2 and R), one instance of the *tank_wheel* class (T), one instance of the *mixer* class (Mix) and one instance of the *button* class (B).

The industrial controller can be specified by the shobi-PN in Fig. 3. The design is a concurrent Moore machine comprising 23 places and 13 transitions. This shobi-PN possesses several closing paths and must use the replica mechanism. This need is imposed by the existence of concurrent invokations of different methods to the same object (the invokation of WR_open(H), RD_full(H) and RD_full(L) to the central reactor R). Whenever the object R arrives to transition t4, it is divided into two replica, each one following different paths until they arrive to transition t11, where they are unified again. An alternative to the replica mechanism would be to decompose the reactor R in several sub-objects, since the concurrent methods' invokations are made to distinct parts.

The shobi-PN has no initially marked control places, but it has two declarations of reinitializing nodes (t1 as an initial node and t13 as a final node. The control net is then reinitialized by adding one marked place, which is an input place to t1 and an output place to t13. To obtain the structure of the associated net (Fig. 4.), it is necessary to remove all the closing places (pc1, pc2, ... pc8) and the arcs that are connected to these places.

For the interpretation of the associated net, the colours references must be deleted and the methods invokations must be transformed into hardware signals.

The textual specification of the system controller in ConPar notation is listed in Fig. 6. As an example of a generated code for synthsizing the controller, a data flow VHDL file (Fig. 7.) was produced by an already implemented CUM. The ASSERT statements, automatically generated by the compiler tool, help the user in the system simulation. Those statements roughly detect transitions in conflict and deadlock situations.

V. CONCLUSIONS

This communication shows that the shobi-PN model is an useful and efficient modelling tool to specify industrial digital control systems. This model is the only known formalism using object-oriented PNs to specify both the parallel control unit and the plant in an integrated and modular way. The shobi-PN model presents synchronous behaviour, object-oriented approaches, and hierarchical mechanisms. As a consequence, this new approach directly supports hierarchical structures in both the control unit and the plant, allowing the specification of industrial digital parallel control systems in a modular, hierarchical and incremental way.

A complete CAD environment was presented, SOFHIA, which supports the specification, analysis, animation, simulation and synthesis of digital systems. The environment generates a file with the control unit specification. The SOFHIA environment can be used for Hw/Sw Codesign which allows the system to be implemented in several technologies.

The analysis of some case studies —included in this communication is the Industrial Reactor— also shows that there is a relation between the structure of the PNs and the kind of approach followed in the system specification. In the Reactor example, the obtained PN reflects a control-driven approach, because the control net is open. On the other hand, in the data-driven approach the nets embody in the control net the reinitializations of the objects with no need to partially or totally incorporate the closing tracks. The data-driven control nets can be totally closed (i.e. they do not have reinitializing nodes). On the other hand, the control-driven control nets must be opened. This analysis shows that the shobi-PN directly follows a data-driven approach in the specification of parallel digital systems, and it properly supports the specification of both the control unit and the plant.

VI. REFERENCES

[1] Luís Gomes and A. Steiger-Garção. Programmable Contoller Design Based on a Synchronized Colored Petri Net Model and Integrating Fuzzy Reasoning. In G. De Michelis and M. Diaz, editors, *Applications and Theory of Petri Nets 1995*, volume 935 of *Lecture Notes in Computer Science*, pages 218–237. Springer-Verlag, Berlin, Germany, 1995.

[2] Manuel Silva. Logical Controllers. In A. De Carli, editor, *IFAC Low Cost Automation: Techniques, Components and Instruments, Applications*, volume II, pages F157–F166bis, Milan, Italy, November 1989.

[3] René David and Hassane Alla. *Petri Nets & Grafcet; Tools for modelling discrete event systems*. Prentice-Hall International, UK, 1992. ISBN 0-13-327537-X.

[4] James Pardey and Martin Bolton. Logic Synthesis of Synchronous Parallel Controllers. *Proceedings of the IEEE International Conference on Computer Design*, pages 454–7, 1991.

[5] R. Valette, M. Courvoisier, J.M. Bigou, and J. Albukerque. A Petri Net Based Programmable Logic Controller. In *IFIP First International Conference on Computer Applications in Production and Enginnering*, April 1983.

[6] João M. Fernandes, António M. Pina, and Alberto J. Proença. Concurrent Execution of Petri Nets based on Agents. In *1st Workshop on Object-Oriented Programming and Models of Concurrency within the XVI International Conference on Applications and Theory of Petri Nets*, Torino, Italy, June 1995.

[7] Tadao Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–80, April 1989.

[8] Kwang-Hyung Lee and Jöel Favrel. Hierarchical Reduction Method for Analysis and Decomposition of Petri Nets. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-15(2):272–80, 1985.

[9] Charles Lakos. The Object Orientation in Object Petri Nets. In *1st Workshop on Object-Oriented Programming and Models of Concurrency*, Torino, Italy, June 1995.

[10] Kurt Jensen. An Introduction to the Theoretical Aspects of Coloured Petri Nets. Technical report, Comp. Science Dept, Aarhus University, Denmark, August 1994.

[11] Ricardo J. Machado, João M. Fernandes, and Alberto J. Proença. SOFHIA: A CAD Environment to Design Digital Control Systems. In *Proceedings of the XIII IFIP Conference on Computer Hardware Description Languages and Their Applications (CHDL'97)*, Toledo, Spain, April 1997. Chapman & Hall.

[12] João M. Fernandes, Marian Adamski, and Alberto J. Proença. VHDL Generation from Hierarchical Petri Net Specifications of Parallel Controller. *IEE Proceedings: Computers and Digital Techniques*, 1997. To appear.

[13] Marian Adamski. Direct Implementation of Petri Net Specification. In *7th International Conference on Control Systems and Computer Science CSCS7*, pages 74–85, 1987.

[14] Marian Adamski and João L. Monteiro. PLD Implementation of Logic Controllers. In *Proceedings of the IEEE International Symposium on Industrial Electronics ISIE'95*, pages 706–11, Athens, Greece, 1995.

```
CLASS: tank
VAR. INST.: BOOL: open
             FLOAT: capacity
CONST.: CAP = xxx
METDS. INST.:
bool RD_OPEN (BOOL level)                         void WR_OPEN (BOOL level)
{ if (level == HIGH)                              { if (level == HIGH)
  then return (open)                                then open = HIGH
  else return (NOT open)                            else open = LOW
}                                                 }
float RD_CAP ()                                   void WR_CAP (FLOAT value)
{ return (capacity)                               { capacity = value
}                                                 }

CLASS: tank_sensor
SUBCLASS OF: tank
VAR. INST.: BOOL: full, empty
METDS. INST.:
bool RD_FULL (BOOL level)                         void WR_FULL (BOOL level)
{ if (level == HIGH)                              { if (level == HIGH)
  then return (full)                                then full = HIGH
  else return (NOT full)                            else full = LOW
}                                                 }
bool RD_EMPTY (BOOL level)                        void WR_EMPTY (BOOL level)
{ if (level == HIGH)                              { if (level == HIGH)
  then return (empty)                               then empty = HIGH
  else return (NOT empty)                           else empty = LOW
}                                                 }
void WR_CAP (FLOAT value)
{ if (value >= CAP)
  then full = HIGH
  else full = LOW
  if (value == 0)
  then empty = HIGH
  else empty = LOW
  capacity = value
}

CLASSE: tank_wheel
SUBCLASS OF: tan
VAR. INST.: BOOL: go, back, empty, start, end
METDS. INST.:
bool RD_GO (BOOL level)                           void WR_GO (BOOL level)
{ if (level == HIGH)                              { if (level == HIGH)
  then return (go)                                  then go = HIGH
  else return (NOT go)                              else go = LOW
}                                                 }
bool RD_BACK (BOOL level)                         void WR_BACK (BOOL level)
{ if (level == HIGH)                              { if (level == HIGH)
  then return (back)                                then back = HIGH
  else return (NOT back)                            else back = LOW
}                                                 }
bool RD_EMPTY (BOOL level)                        void WR_EMPTY (BOOL level)
{ if (level == HIGH)                              { if (level == HIGH)
  then return (empty)                               then empty = HIGH
  else return (NOT empty)                           else empty = LOW
}                                                 }
void WR_CAP (FLOAT value)
{ if (value == 0)
  then empty = HIGH
  else empty = LOW
  capacity = value
}
```

Fig. 5. The classes for the industrial reactor.

```
<*---- Input and Output signals ----*>
.clock  CLK
.input  START FULLC1 EMPTYC1 FULLC2 EMPTYC2 FULLR EMPTYR BEGIN
        END EMPTYT
.output OPENS1 OPENS2 OPENC1 OPENC2 OPENR TURN GO BACK OPENT

<*---- Industrial Reactor SIPN ----*>
.part REACTOR
.place p1 p2 p3 p4 p5 p6 p7 p8 p9 p10 p11 p12 p13 p14 p15 p16
.transition t1 t2 t3 t4 t5 t6 t7 t8 t9 t10 t11 t12 t13

.net
t1:  p1  *       START    |- p2 * p3 * p6;
t2:  p2  *       FULLC1   |- p4;
t3:  p3  *       FULLC2   |- p5;
t4:  p4  * p5             |- p8 * p9 * p10;
t5:  p8  *       FULLR    |- p7;
t6:  p7  *      !FULLR    |- p8;
t7:  p9  *       EMPTYC1  |- p11;
t8:  p10 *       EMPTYC2  |- p12;
t9:  p6  *       BEGIN    |- p13;
t10: p11 * p12 * p13      |- p14;
t11: p8  * p14 * EMPTYR   |- p15;
t12: p15 *       END      |- p16;
t13: p16 *       EMPTYT   |- p1;

.MooreOutput
p2   |-  OPENS1;
p3   |-  OPENS2;
p6   |-  BACK;
p7   |-  TURN;
p9   |-  OPENC1;
p10  |-  OPENC2;
p14  |-  OPENR;
p15  |-  GO;
p16  |-  OPENT;

<*---- Initial Marking ----*>
.marking p1
.e
```

Fig. 6. ConPar code for the industrial reactor.

```
ENTITY controller IS
  PORT (reset, start, fullc1, ..., end, emptyt, clk : IN BIT;
        opens1, opens2, openc1, ..., back, opent : OUT BIT);
END controller;

ARCHITECTURE dataflow OF controller IS

-- Place Signals
  SIGNAL p1   : REG_BIT REGISTER;
  SIGNAL Np1  : BIT;
  SIGNAL p2   : REG_BIT REGISTER;
  SIGNAL Np2  : BIT;
  ...
  SIGNAL p16   : REG_BIT REGISTER;
  SIGNAL Np16 : BIT;
-- Transition Signals
  SIGNAL t1 : BIT;
  SIGNAL t2 : BIT;
  ...
  SIGNAL t13 : BIT;

BEGIN
  PART : BLOCK (clk='1' AND NOT clk'STABLE)
  BEGIN
    p1  <= GUARDED Np1  WHEN reset='0' ELSE '1';
    p2  <= GUARDED Np2  WHEN reset='0' ELSE '0';
    ...
    p16 <= GUARDED Np16 WHEN reset='0' ELSE '0';
  END BLOCK;

-- Dataflow description for transitions
  t1 <= NOT p2 AND NOT p3 AND NOT p6 AND start AND p1;
  t2 <= NOT p4 AND fullc1 AND p2;
  t3 <= NOT p5 AND fullc2 AND p3;
  t4 <= NOT p8 AND NOT p9 AND NOT p10 AND p5 AND p4;
  t5 <= NOT p7 AND fullr AND p8;
  t6 <= NOT p8 AND NOT fullr AND p7;
  t7 <= NOT p11 AND emptyc1 AND p9;
  t8 <= NOT p12 AND emptyc2 AND p10;
  t9 <= NOT p13 AND begin AND p6;
  t10 <= NOT p14 AND p13 AND p12 AND p11;
  t11 <= NOT p15 AND emptyr AND p14 AND p8;
  t12 <= NOT p16 AND end AND p15;
  t13 <= NOT p1 AND emptyt AND p16;

-- Dataflow description for next place markings
  Np1 <= t13 OR (p1 AND NOT t1);
  Np2 <= t1 OR (p2 AND NOT t2);
  Np3 <= t1 OR (p3 AND NOT t3);
  Np4 <= t2 OR (p4 AND NOT t4);
  Np5 <= t3 OR (p5 AND NOT t4);
  Np6 <= t1 OR (p6 AND NOT t9);
  Np7 <= t5 OR (p7 AND NOT t6);
  Np8 <= t6 OR t4 OR (p8 AND NOT t11 AND NOT t5);
  Np9 <= t4 OR (p9 AND NOT t7);
  Np10 <= t4 OR (p10 AND NOT t8);
  Np11 <= t7 OR (p11 AND NOT t10);
  Np12 <= t8 OR (p12 AND NOT t10);
  Np13 <= t9 OR (p13 AND NOT t10);
  Np14 <= t10 OR (p14 AND NOT t11);
  Np15 <= t11 OR (p15 AND NOT t12);
  Np16 <= t12 OR (p16 AND NOT t13);

-- Output Signals Equations
  opens1 <= p2;
  opens2 <= p3;
  openc1 <= p9;
  openc2 <= p10;
  openr <= p14;
  turn <= p7;
  go <= p15;
  back <= p6;
  opent <= p16;

-- Transitions in conflict
  ASSERT NOT (t6 AND t4)
    REPORT "t6 and t4 are in conflict, because of output place p8."
    SEVERITY ERROR;
  ASSERT NOT (t11 AND t5)
    REPORT "t11 and t5 are in conflict, because of input place p8."
    SEVERITY ERROR;
-- No Enabled Transitions
  ASSERT NOT (t1='0' AND t2='0' AND ... t13='0')
    REPORT "Petri Net may be deadlocked"
    SEVERITY WARNING;

END dataflow;
```

Fig. 7. VHDL code for the industrial reactor.