# A TRANSCRIPTION TOOL FROM PETRI NET TO CLP PROGRAMMING LANGUAGES

**André Torres Ferraz de Mello, andre.torres.mello@gmail.com**
**Marcelo Castro Barbosa, celocb@gmail.com**
**Diolino José dos Santos Filho, diolinos@usp.br**
**Paulo Eigi Miyagi, pemiyagi@usp.br**
**Fabrício Junqueira, fabri@usp.br**
Escola Politécnica da Universidade de São Paulo, São Paulo, SP, Brazil

***Abstract.*** *Petri net is a powerful graphic tool for discrete event systems modeling. However, translation from Petri nets to programmable logic controller (PLC) programming language, in order to provide dynamic project integration, is still done manually, given the unavailability of translators to compile different types of Petri nets into direct and specific programming languages. The goal of this paper is to address this drawback by developing and implementing a comprehensive translator, capable of interpreting most used Petri net and translating them to PLC-ready language. To achieve this goal, an extensive research was made, in order to evaluate the many extensions of the original Petri net language, their characteristics and modeling power. The IEC 61131-3 PLC programming languages were also studied. After that, the inputs and outputs of the translator were chosen: as the input, the software supports SIPN (Signal Interpreted Petri Nets), which is a type of Petri net that assigns input variables to transitions and output variables to places. This input net must be written in the PNML (Petri Net Markup Language), a recently established interchange standard. For the output, the program compiles the net into three different PLC languages: ladder diagram, sequential function chart and structured text, through specific algorithms. The output is written in the PLCOpen XML language, a format that is a trend on the standardization of the PLC programming languages and their interchange. Beside the translation process itself, the program also supports stepwise simulation of the chosen Petri net, allowing for error-checking during the development. In this way, it creates a seamless development workflow for PLCs programming, using Petri net as the base starting language.*

***Keywords****: Petri net; Programmable logic controller; Transcription tool*

## 1. INTRODUCTION

Programmable logic controllers (PLC) are powerful tools in the automation area. It is being applied in small and simple machines as well as large manufacturing plants (FREY, 2001). PLC are widely used and distributed for industrial automation. PLC is used in applications that need to perform logical operations, sequencing, timing and numerical computation. It has a memory where control procedures are written in the form of a list of commands. Based on the content of this memory, the machine operations and/or processes are controlled by digital and/or analogical output signals (MIYAGI, 1996). However, there is a lack of formal methods for specification, validation and verification of code generated, that minimizes the chances of design errors.

There is a common separation between the programming languages: textual and graphic. There are also numerous discussions that point out the advantages and disadvantages of both due to differences in the way of programming. In the case of computer programming languages and mathematical notation, the textual language predominates (Barros, 2006). Despite the graphic languages have already conquered their space (OMG, 2003), they are often viewed as inherently less accurate than textual languages. However, Harel and Rump (2004) apud (Barros, 2006) claim that this view is not true. Specifically, the weaknesses are the difficulty of enforcement and implementation of language and inability to graphically describe sequential and concurrent algorithms and visual analysis on the evolution of states (current state, next state, etc.).

Petri net is just a graphical tool that has strong attributes for modeling and control of discrete event systems. According Valk (2003) apud (Barros, 2006), the comparative advantages of Petri nets over other languages consist of simple graphical representation (circles, rectangles and arcs), simple algebraic representation (in low level Petri net), structured algorithms based on the duality of place-transition and, finally, local (well defined) effect of transitions. Thus, the Petri net model allows modeling and validation of control algorithms through several techniques including simulation. However, translation from Petri net to PLC programming language is still done manually, given the unavailability of translators to compile different types of Petri nets into direct and specific programming language.

Therefore, this work proposes a transcription tool from Petri net to PLC programming languages, so that the process of design and implementation becomes more efficient and less vulnerable to errors.

Some concepts that support this work are presented in Section 2. Section 3 presents the procedure to transcribe Petri net for IEC 61131-3 programming languages. Section 4 presents a workflow for modeling and implementation of the PLC control logic. Section 5 presents an application example, and section 6 presents the main conclusions.

## 2. BACKGROUND

We consider that the basic ideas and nomenclature used in Petri net texts are well known, but if detailed information is need, see Miyagi (1996) for example.

### 2.1. SIPN

Signal Interpreted Petri net (SIPN) (FREY, 2001) is an extension of Condition Event Petri net (MIYAGI, 1996). It has been developed to support the design control logic algorithms for Programmable Logic Controllers (PLC). Hence they have to be connected to the plant under control and that is achieved via the handling of signals. For this, the inputs and outputs that represent this interface must be implemented in Petri net.

SIPN transitions are associated with firing conditions, Boolean functions of the input variables of the net. They are evaluated after the transition has been enabled (input places of the transition are all marked and output places are all unmarked) and produce the immediate firing of the transition.

SIPN places are associated with output functions. An output function is a piece of PLC code that is executed when the considered place is marked. It affects the output signal of the PLC.

The second important point in the description of a SIPN is the way it behaves. Opposite to ordinary Petri net, an immediate and simultaneous firing of transitions that can fire always occurs. Moreover, the firing process is realized until no more transition can fire under this input setting. The reached marking is said to be stable. After such a marking has been reached, the output functions are evaluated and given out.

To handle real problems, the SIPN has been extended with time and hierarchy concepts. Hence, every arc from a place to a transition can be associated with a time delay. This delay represents the minimal time a token has to spend in the input place before the transition is enabled. Furthermore, as real industrial systems tend to be composed for a great number of equipments, machines, processes, etc., the concept of hierarchy has been introduced. It allows replacing an identified quite independent partition of a SIPN by a single hierarchical place. The extension of this place (subnet) contains the description of the identified process part.

### 2.2. PNML

In November 2009, the ISO (International Organization for Standardization) published the ISO / IEC 15909-2, which describes a transfer format for Petri net based XML (eXtended Markup Language). This format, the Petri net Markup Language (PNML), enables the exchange of Petri nets between different tools and different working groups. The standard also defines concepts about the graphical appearance of the Petri net and its syntax (ISO 2010).

Among its key concepts can be highlighted (Kindle and Weber, 2003):

- Principles of design: the PNML and its use are guided by the goals of flexibility, compatibility and disambiguation;
- Petri net type: there are several types and extensions of Petri net. The PNML takes this into account when representing each Petri net and decide which is your constitutional type;
- PNML structure: the language has some modules. They are:
  - Meta model: the file containing the Petri net itself;
  - Type definition interface: an interface for defining Petri net types;
  - Feature definition interface: module where new features are defined, exclusive of certain tools and Petri net types;
  - Conventions document: since several Petri net types are widely used, they can be standardized. This is the purpose of the document of conventions, a standardization system that is gradually updated by users and developers;
  - Petri net and objects: each PNML file may contain one or more Petri net. Both the Petri net and its elements are called objects. There are four types of objects in the basic version of the PNML language: arc; place; transition; and net.
- Labels: every object, including the Petri net itself may have annotations. These annotations can represent names, comments, variables, conditions, etc., depending on the Petri net type;
- Graphic information: for graphic display of the Petri net, the PNML reserves specific information such as coordinates of exhibition, positions relative to other objects, etc.;
- Tools information: some Petri net editors and simulators have additional features. Thus, it is possible to record additional information. When the file is opened by other than the original tool, the information is ignored and the Petri net is normally opened.

### 2.3. IEC 61131-3 and the PLCOpen

The IEC 61131-3 standardizes five programming languages for PLC: Sequential Function Chart (SFC), Ladder Diagram (LD), Function Block Diagram (FBD), Instruction List (IL) and Structured Text (ST). Table 1 presents a set of classifications that can be applied to those languages (MIYAGI, 1996).

Such diversity of programming languages can be attributed to competition among manufacturers that sought to find a simpler or more efficient ways to program their products, differentiating their products, and/or acquire market advantage (Warner et al., 2006) apud (SARMENTO, 2008). However, this diversity hinders the exchange of programs between equipment from different manufacturers, and raises the costs of personnel training and maintenance. Thus, the PLCopen (PLCopen, 2008), an organization whose mission is to solve problems related to use of PLC in order to achieve a level of international standardization, constituted a technical committee called TC6 to draft a language based on XML – the PLCOpen XML[1] – that supports the exchange of information written in the IEC 61131-3 languages.

Table 1. Classification of languages used in PLC programming (Miyagi, 1996).

| **Type** | **Languages** | **Features** | | |
|---|---|---|---|---|
| | | **Logic** | **Ordination** | **Complex functions** |
| **Textual** | Boolean algebra | X | | |
| | IL | X | | |
| | ST | X | | |
| **Graphics** | LD | X | | X |
| | FBD | X | | X |
| | Flowchart | | X | X |
| | SFC | | X | |
| **Tabular** | Decision table | | X | |

Therefore, in order to use the PLCopen XML, three IEC 61131-3 programming languages were chosen in this study: LD, SFC, and ST.

The LD was chosen because it is the PLC programming language most widely adopted in industry (Lucas and Tilbury, 2005). Its application is varied and flexible (Lucas and Tilbury, 2005), and the design concept is derived directly from the electromagnetic relay (FREY, 2001), making it an easy to use graphic language (KARL-HEINZ and TIEGELKAMP, 2001).

The SFC, that it is suitable for modeling of processes that can be divided into steps (TIEGELKAMP and KARL-HEINZ, 2001). It is the language that most closely resembles the Petri net.

Finally, the ST was chosen because it is the textual language of IEC61131-3 with the largest growth in terms of usage and acceptance. Furthermore, its understanding is simple for users familiar with high-level programming languages (Thayer, 2009).

More details about these languages can be found in Miyagi (1996).

## 3. PROCEDURE TO TRANSCRIBE PETRI NET FOR IEC 61131-3 PROGRAMMING LANGUAGES

This section presents the procedures used by the transcription tool to translate Petri Net for each programming languages of IEC 61131-3 selected for this work.

### 3.1. Transcription from Petri net to LD

The method chosen to transcribe Petri net for LD is based on Santos Filho and Miyagi (1997). This method divides the LD in three blocks: (1) determine which transitions are enabled; (2) firing of transitions, removing the tokens from input places and creating tokens on the output places; and (3) activate the outputs corresponding to the places with tokens. Additionally, it should generate an additional block that sets the initial conditions of the system. This block corresponds to the initial marking of Petri net.

The transcription process can be organized into the following steps:

**A) Create auxiliary variables**

These variables represent the state of the elements of the Petri net (places and transitions). Thus, each Petri net place has two associated variables: (a) the output variable of the program, represented by the name of the place, (b) the auxiliary variable that represents the presence of a token on it. In this work, the auxiliary variable is named using the place name followed by the termination "Local". So, the place L1 is associated with the variables "L1" and "L1Local".

The same happens with transitions: (a) the input variable of the program, represented by the name of the transition (e.g. "T1"), (b) the auxiliary variable that represents the enabling of the transition (e.g. "T1Local").

[1] The syntax of this language can be found at http://www.plcopen.org/pages/tc6_xml/index.htm.

**B) Enabling of transitions**

Each transition is defined as a "coil" and is associated with a rung of LD. This coil is identified with the name of the local variable (e.g., "T1Local"). The input places (pre conditions) are set as normally open contacts, and the output places (pos conditions) as normally closed contacts. Finally, on the same rung, the input variables associated with the transitions are defined as contacts.

**C) Firing of transitions**

In this section, for each transition is created so many LD rungs as arcs connected to it. For input arcs, the rung has a reset coil. For output arcs, the rung has a set coil. Each coil is named whit the name of the auxiliary variable associated with each place (e.g. "P1Local"). For each rung is created a normally open contact corresponding to the transition and named with the name of the auxiliary variable (e.g. "T1Local").

**D) Activating the outputs**

A LD rung is associated with each output variable. This variable is represented by a common coil. Each (Petri net) place associated with the output variable is represented by a normally open contact and named with the name of the auxiliary variable (e.g. "P1Local"). If more than one place is associated with an output variable, the contacts are put in rung in parallel.

**E) Initial conditions**

The initial conditions of the program are defined with the aid of a LD rung. In this rung, each place is represented by a normally closed contact (named with the name of the auxiliary variable associated with the place). Thus, the coils in this rung will be accessed only during the first cycle of the PLC when all contacts are false. These coils are of type "set". Each coil corresponds to a place with initial marking.

3.2. Transcription from Petri net to ST

The method chosen to transcribe Petri net for ST is based on item 3.1 method, organizing the program in three blocks plus the initial conditions. The transcription process can be organized into the following steps:

**A) Create auxiliary variables**

The same described in item 3.1.

**B) Enabling of transitions**

For each scantime, the first step is to define which transitions are enabled. For this, a logical combination (AND) of three conditions must be performed: (a) existence of tokens in the input places (auxiliary variables are "TRUE"); (b) absence of tokens in the output places (auxiliary variables are "FALSE"); and (c) input variables associated with the transition are satisfied ("TRUE").

**C) Firing of transitions**

To simulate the dynamic of tokens in the Petri net transcribed, it is necessary the use of set and reset structures to update the local variables associated with places. In ST, the structure used to represent this statement is the "IF". Each transition is associated with an "IF". Inside each statement, "TRUE" is assigned to all the local variables associated with output places, and "FALSE" for all local variables associated with input places.

It is important to note that the IF statement behaves like a set or reset coil. It does not have an "ELSE" clause.

**D) Activating the outputs**

The value of auxiliary variable associated with each place is assigned to its respective output variable (e.g. P1 := P1Local).

**E) Initial conditions**

To set the initial condition of the program (initial marking in the Petri net) it is necessary to create a statement that will be executed only once by the PLC. This condition corresponds to the moment where all places are unmarked. Thus, the program verifies ("IF") if all the auxiliary variables associated to places are "FALSE". If this condition is met, "TRUE" must be assigned to all the auxiliary variables corresponding to the places initially marked.

3.3. Transcription from Petri net to SFC

The method chosen to transcribe Petri net for SFC is based on Santos Filho and Miyagi (1997) too. However, this case there is an explicit similarity between the input and output languages of the transcriber. This is because the SFC is derived from Petri net (Miyagi, 1996). This similarity simplifies the transcription process since a relationship between

the different structural elements of the two languages must be established. Thus, Table 2 shows the mapping between the two languages.

Table 2. Mapping between Petri net and SFC.

| **Petri net** | **SFC** |
|---|---|
| Place | Step |
| Transition | Transition |
| Arc | Connection |
| Input variables | Receptivity of each transition |
| Output variables | Action of each step |

The particularity is related to situations of conflict and parallelism. The cases of conflict are not addressed in the transcription tool. However, the tool notifies the user of the existence of conflicts in the Petri net.

Petri net parallelism occurs when two or more arcs leave the same transition. When this transition fires, all output places receive tokens, characterizing the parallelism.

In the transcription process is introduced a parallel line (double line) in the SFC diagram. The parallel line is connected to the transition. Thus, instead of the connections leave the transition to the steps, they leave the line of parallelism. The same occurs with the synchronization (end of parallelism).

Some care is necessary in the construction of the Petri net that will be converted into SFC:

- The Petri net must have only one place marked (initial condition);
- For systems that perform repetitive operations, the Petri net must have only one arc that points to the initial condition. With this practice, the transcription tool is able to identify the connection corresponds to this arc, replacing it by a "jump" for the initial condition.

## 4. WORKFLOW FOR MODELING AND IMPLEMENTATION

A workflow is proposed in this section in order to guide users in the design and implementation of control algorithms for PLC. This process starts with the modeling of the control algorithm using a Petri net editor and ends with the verification of the generated code. The workflow is shown in Figure 1 and its description is given following.

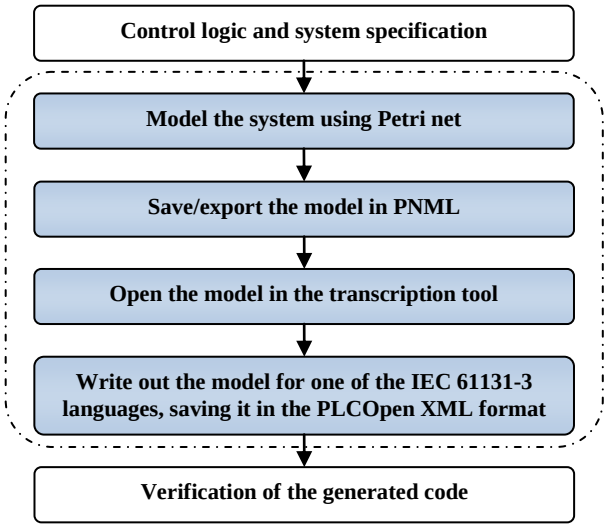



Figure 1. Workflow for modeling and implementation

**Step 1: Model the system using Petri net**

The control system should be modeled and simulated using a Petri net editor. It is suggested that the editor allows the entry of input variables in transitions, and output in places, as provided by SIPN. For example, NetLab[2] can be used. Entries are made in "labels" of the elements. Thus, for the transcription tool, the name of a place is the name of the output variable, while the name of a transition corresponds to the input variable.

However, the use of auxiliary variables may be needed. Thus, to differentiate these variables from inputs and outputs, the auxiliary variables receive the word "default" in its name (e.g. "defaultPlace1", "defaultTransition2",

[2] http://www.irt.rwth-aachen.de/en/fuer-studierende/downloads/petri-net-tool-netlab/

"defaultVariable3", etc.). To use a negated variable or entry, it is used "!" before the variable name, e.g. "!place5" or "!transition4".

**Step 2: Save/export the model in PNML**

As previously mentioned, the tool used for modeling and simulation of the system must be compatible with the ISO/IEC 15909-2 PNML standard, the input format used by the transcription tool.

**Step 3: Open the model in the transcription tool**

The model can be opened in the transcription tool. To ensure that the PNML standard is being followed by the modeling tool, the transcription tool allows the visualization of Petri net.

**Step 4: Write out the model for one of the IEC 61131-3 languages, saving it in the PLCOpen XML format**

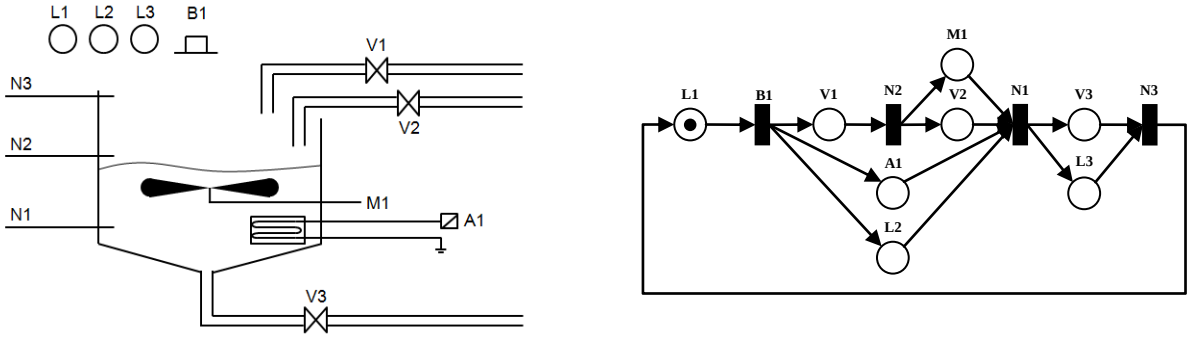
As previously stated, the Petri net can be transcribed into three languages provided by IEC 61131-3: LD, SFC, and ST.

Despite efforts of PLCOpen to standardize the file exchange between different languages, the adoption of this standard in commercial tools is still slow. Thus, to validate the generated codes were used two tools: the Beremiz[3] and CoDeSys[4].

## 5. EXAMPLE

A mixer tank (Figure 2(a)), generally used in chemical industry, was used to illustrate the use of the transcription tool. The proposed mixer has two fluid inputs, for solvent and for solute, controlled through the valves V1 and V2 respectively. The final mixture is drained by the valve V3. The mixing process still needs a heater and a mixer, that are activated by relays A1 and M1, respectively.

There are three sensors for the detection of liquid level in the tank: N1, N2 and N3. There are three lamps (L1, L2 and L3) to monitor the process. The lamp L1 denotes that the system is ready, awaiting the start command. The lamp L2 indicates that the reservoir is filling and the lamp L3 indicates that it is draining. Finally, the button B1 is used to initialize the process.




(a) (b)

Figure 2. (a) Example of mixing system (based on Miyagi (1996)),
(b) Petri net describing the control logic of the mixer system.

Once initialized, the valve V1 is opened to enable entry of solvent. Simultaneously the heater A1 is turned on to ensure constant temperature along the process. The lamp L2, indicating the "tank filling", is turned on too.

The valve V1 is closed when the solvent rises to the sensor N2. Concomitant, the valve V2 is opened enabling entry of solute, and the mixer M1 is activated.

When the mixture reaches sensor N3, the valve V2 is closed, the mixer M1, the heater A1, and the lamp L2 are turned off. Then, the valve V3 is opened to empty the tank. The lamp L3, indicating the "tank emptying", is turned on too.

Finally, when the mixture rises to the sensor N1, the valve V3 is closed and the lamp L3 is turned off. The process returns to its initial state, which is signaled by the lamp L1, lit.

Based on the description of the process, and the input and output signals, the process control was generated and represented using SIPN (Figure 2 (b)). The input and output variables were mapped in the transitions and the places,

[3] http://www.beremiz.org/
[4] http://www.3s-software.com/

respectively. Thus, each place has the name of its associated output variable, and each transition, the name of its associated input variable.

5.1. The resulting LD

The resulting LD is divided in four sections as showed in Figure 3. The first block (Figure 3(a)) is equivalent to the firing of each transition according to the tokens in the input places and its input variable(s). The second block (Figure 3(b)) represents the flow of tokens through the Petri net. For each transition, each arc connected to the transition generates a rung in the LD diagram. Input arcs generate rungs with reset coil (removes the token from the correspondent place); and output arcs generate rungs with set coil (inserts a token in the correspondent place). The third block (Figure 3(c)) is used to set the output variables according to the existence of tokens in the corresponding place. For each output variable is created a rung. The coil represents the output variable. Finally the initial conditions are created for the system (Figure 3(d)), i.e., in this case a token is created in place L1. If no place has tokens (all coils are reset), the coil L1 is activated.

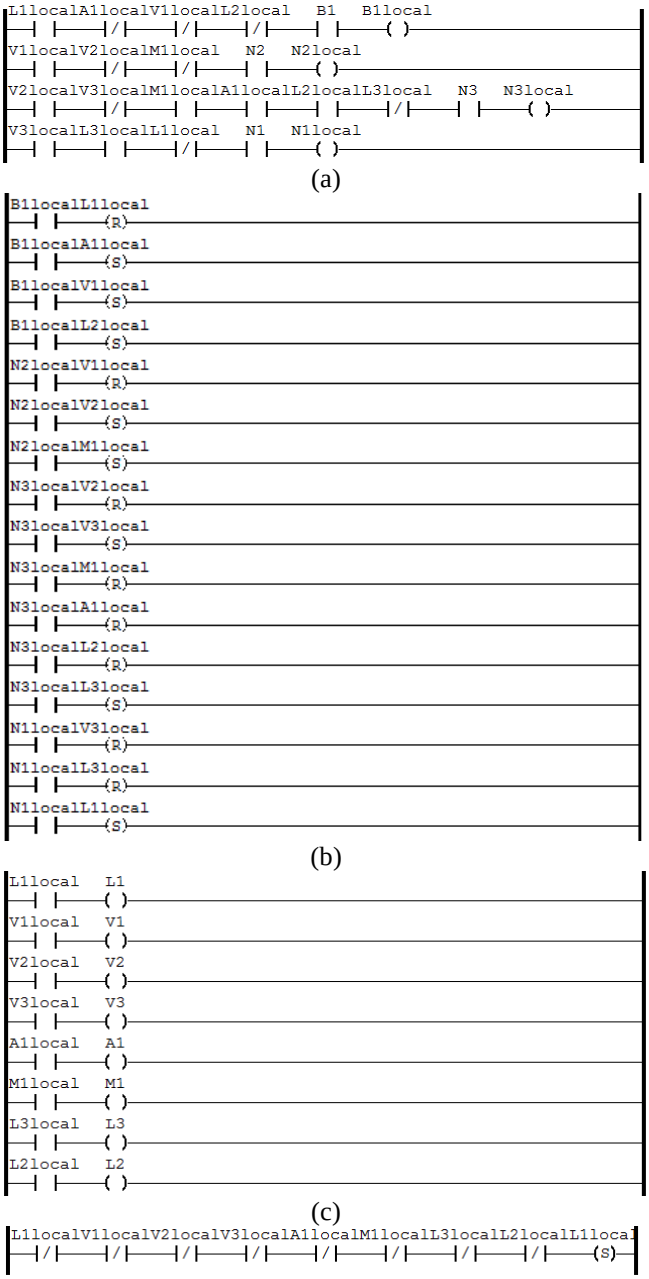



Figure 3. LD diagram generated by transcription tool and displayed in Beremiz: (a) block corresponding to transition firing, (b) block representing the flow of tokens, (c) block representing the activation of the output variables, and (d) block to initialize the initial marking.

5.2. The resulting SFC

There is a great similarity between the SFC (Figure 4) and Petri net (Figure 2(b)). It is noticed that each element of the Petri net has been converted on the structural equivalent of the SFC. Each step is named with the corresponding place name plus the word "Local". In this example, as the name of each place matches the output variable, it became the name of the action block with the N qualifier, which means that while the `step` is active, the output will be also.

In this example, the transition names are the input variables. After the transcription, the name of each Petri net transitions were converted in the receptivity of the SFC `transitions`, arranged in the form of structured text.

Finally, it was necessary to include the object "jump" to allow the cycle starts again. This "jump" corresponds to the arc connecting the transition N1 to the place L1.

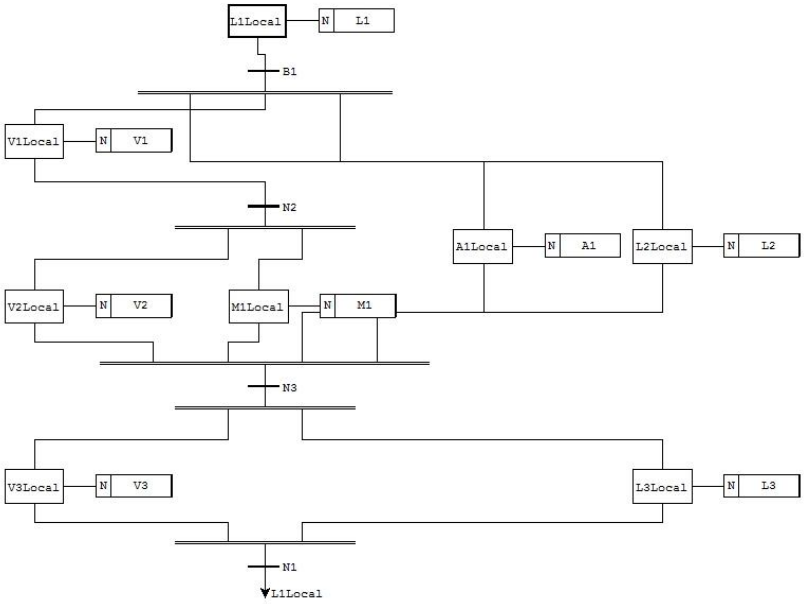



Figure 4. SFC diagram generated by transcription tool and displayed in Beremiz.

5.3. The resulting ST

Similarly to the transcription for LD, the resulting code can be divided into four blocks (Figure 5): (a) block corresponding to transition firing; (b) block corresponding to flow of tokens; (c) block corresponding to the setup of output variables; and (d) the block defining the initial conditions of the system. To verify the transcription of the Petri net to ST, the PLCopen XML file was opened in CoDeSys.

## 6. CONCLUSIONS

For the design of automated systems, modeling and simulation techniques are very useful to validate and verify the control logic. In this sense, Petri net has been used in several studies. However, Petri net is not language provided by IEC 61131-3 for PLC programming.

To minimize coding errors, it is desirable to use tools to automatically transcribe the logic in Petri net in the languages of IEC 61131-3. However, there is no knowledge of the availability of these tools. Thus, this work presented a transcription tool from Petri net for three of the IEC 61131-3 programming languages: LD, SFC and ST.

This tool uses two standards for data exchange: the PNML as input format, which describes the Petri net, and the PLCOpen XML as output format, which describes the program to be loaded into the PLC. These standards were chosen to take a transcription tool independent of manufacturers.

The NetLab was used to generate the Petri net, since it uses the PNML standard. Two programs were used to verify the control logic transcribed: Beremiz, to validate the programs in LD and SFC; and the CoDeSys to validate the program in ST. These programs meet the PLCopen XML for the respective programming languages.

## 7. ACKNOWLEDGEMENTS

The authors would like to thank the Brazilian governmental agencies CAPES, CNPq and FAPESP for their partial support.

```
B1Local := L1Local AND NOT A1Local AND NOT V1Local AND NOT L2Local AND B1;
N2Local := V1Local AND NOT V2Local AND NOT M1Local AND N2;
N3Local := V2Local AND NOT V3Local AND M1Local AND A1Local AND L2Local AND NOT
L3Local AND N3;
N1Local := V3Local AND L3Local AND NOT L1Local AND N1;
```

(a)

```
IF B1Local = TRUE THEN L1Local := FALSE; END_IF
IF B1Local = TRUE THEN A1Local := TRUE; END_IF
IF B1Local = TRUE THEN V1Local := TRUE; END_IF
IF B1Local = TRUE THEN L2Local := TRUE; END_IF
IF N1Local = TRUE THEN V3Local := FALSE; END_IF
IF N1Local = TRUE THEN L3Local := FALSE; END_IF
IF N1Local = TRUE THEN L1Local := TRUE; END_IF
IF N2Local = TRUE THEN V1Local := FALSE; END_IF
IF N2Local = TRUE THEN V2Local := TRUE; END_IF
IF N2Local = TRUE THEN M1Local := TRUE; END_IF
IF N3Local = TRUE THEN V2Local := FALSE; END_IF
IF N3Local = TRUE THEN V3Local := TRUE; END_IF
IF N3Local = TRUE THEN M1Local := FALSE; END_IF
IF N3Local = TRUE THEN A1Local := FALSE; END_IF
IF N3Local = TRUE THEN L2Local := FALSE; END_IF
IF N3Local = TRUE THEN L3Local := FALSE; END_IF
```

(b)

```
A1 := A1Local;
L1 := L1Local;
L2 := L2Local;
L3 := L3Local;
M1 := M1Local;
V1 := V1Local;
V2 := V2Local;
V3 := V3Local;
```

(c)

```
IF NOT L1Local AND NOT V1Local AND NOT V2Local AND NOT V3Local AND NOT A1Local
AND NOT M1Local AND NOT L3Local AND NOT L2Local THEN
     L1Local := TRUE;
END_IF
```

(d)

Figure 5 ST instructions generated by transcription tool and verified in CoDeSys: (a) block corresponding to transition firing, (b) block representing the flow of tokens, (c) block representing the activation of the output variables, and (d) block to initialize the initial marking.

## 8. REFERENCES

AUTOMATION ALLIANCE. Automation Alliance homepage. Automation Alliance website, 2010. Disponivel em: <http://www.automation-alliance.com/>. Accessed in: June 02, 2010.

BARROS, J. P. M. P. R. E. Modularidade em Redes de Petri. Universidade Nova de Lisboa. Lisboa. 2006.

FREY, G. SIPN, Hierarchical SIPN and Extensions. Kaiserslautern: Institue of Automatic Control - University of Kaiserslautern, 2001.

ISO. International Organization For Standarzation. Information processing -- Text and office systems -- Standard Generalized Markup Language (SGML), 2010. Available in: <http://www.iso.org/iso/catalogue_detail.htm?csnumber=16387>. Accessed in: April 12, 2010.

KARL-HEINZ, J.; TIEGELKAMP, M. Programming industrial automation systems: concepts and programming languages, requirements for programming systems, aids to decision-making tools. 1st ed. Heidelberg, Germany: Springer, 2001.

KINDLER, E.; WEBER, M. The Petri Net Markup Language. Lecture Notes in Computer Science, Berlin, 2003. 124-144.

LUCAS, M. R.; TILBURY, D. M. Methods of measuring the size and complexity of PLC programs in different logic control design methodologies. The International Journal of Advanced Manufacturing Technology, London, p. 436-447, September 2005. ISSN 1433-3015.

MIYAGI, P. E. Controle Programável: fundamentos do controle de sistemas a eventos discretos. 1. ed. São Paulo: Editora Blucher, 1996.

OMG. Unified Modeling Language Specification, version 1.5. Object Modeling Group, 2003. Available in: <http://www.omg.org/cgi-bin/doc?formal/03-03-01>. Accessed in: April 13, 2010.

PLCOPEN. Status T6C. PLCOpen, 2008. Available in: <http://www.plcopen.org/pages/whats_new/tc6/status.htm>. Accessed in: April, 13 2010.

PNML ORGANIZATION. PNML Grammar, version 2009. PNML.org, 2010. Available in: <http://www.pnml.org/version-2009/version-2009.php>. Acessed in: June 14 2010.

SANTOS FILHO, D. J.; MIYAGI, P. E. Proposta de uma ferramenta automática de programação de CPS a partir de modelos MFG. COBEM - Congresso Brasileiro de Engenharia Mecânica. Bauru: [s.n.]. 1997.

SARMENTO, C. A. Modelagem De Programas E Sua Verificação Para Controladores Programáveis. São Paulo: Ed. Rev., 2008.

SMART SOFTWARE SOLUTIONS. CoDeSys Homepage. Smart Software Solutions Website, 2010. Available in: <http://www.3s-software.com/index.shtml?homepage>. Accessed in: June 14 2010.

THAYER, T. Speaking in Tongues: Understanding the IEC 61131-3 Programming Languages. Control Engineering, Chicago, 30 January 2009.

WEBER, M. et al. The Petri Net Markup Language: concepts, technology and tools. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Berlin, Germany, 2003. 483-505.

WITSCH, D.; VOGEL-HEUSER, B. Close integration between UML and IEC 61131-3: New possibilities through object-oriented extensions. IEEE Conference on Emerging Technologies and Factory Automation. Mallorca: IEEE. 2009. p. Article 5347155.

## 9. RESPONSIBILITY NOTICE

The author(s) is (are) the only responsible for the printed material included in this paper.