Control and Plant Modeling for Manufacturing Systems using Statecharts

Raimundo Santos Moura
Federal University of Piaui
Terezina, Brazil
Email: rmoura@dca.ufrn.br

Felipe Cesar Alves do Couto
Federal University of Rio Grande do Norte
Natal, Brazil
Email: felipecouto@dca.ufrn.br

Luiz Affonso Guedes
Federal University of Rio Grande do Norte
Natal, Brazil
Email: affonso@dca.ufrn.br

***Abstract*— Design engineers need to use high level methodologies to facilitate the development, maintenance, and documentation of the industrial control systems. These methodologies must include modeling, formal validation, and code generation to *programmable logic controller (PLC)*. This paper discusses a methodology for plant and control modeling of the manufacturing systems using *UML/Statecharts*, and presents a case study in detail to clarify our ideas.**

I. Introduction

Industrial automation uses concepts of the theory of systems to control machines and processes. In the field of modeling, many attempts have been made to develop notations and semantics to classify and describe the different kinds of systems. Such attempts provide the infrastructure required to solve some difficult problems in engineering, and aim at increasing the productivity, quality, and safety of the system development process.

Currently, controller programming is performed by specifically qualified technicians in one of the five languages defined by IEC-61131-3 [1] standard and who seldom have knowledge of modern software technologies. Furthermore, controllers are often reprogrammed during plant operation life-cycle to fit them to new requirements. These two points lead to the fact that *"in practice there is not a formal description to implement programmable controllers"* [2]. Therefore, the use of higher level methodologies in control programming constitutes a great challenge to be conquered.

Software reusability and composability have been discussed since the 80's, with the use of object-oriented methods [3]. In the Industrial area, the IEC-61499 [4] standard allows reuse of application parts (function block, sub-application) in different applications. Software reuse is a complicated problem and depends not only on the means provided by the modeling language, but also on the overall application structure.

In Computer Science, several models guide the software development process such as the **V-Model** [5], that was developed in Germany to regulate this process and became a classical model to the planning and execution of projects; the **Waterfall Model** [6], that is a sequential software development model in which development is seen as sequence of phases; and the **Spiral model** [7], that is an iterative software development model which combines elements of software design and prototype stages.

In short, an application life-cycle can be divided in three phases: **Modeling - Validation - Implementation** (see Fig. 1). Modeling is phase that demands more time in application life-cycle. The "Modifications" arc represents multiple iterations that can occur in software modeling processes. The "Re-engineering" arc represents the research area, which investigates the generation of a model from legacy code. Our focus are in forward engineering, which investigate the model generation from requirements specified by users.

In literature, there are several approaches that present methodologies, languages, and patterns for modeling industrial applications, especially for *Discrete Event Systems (DES)* [8]. The two most common approaches are *Finite State Machines (FSM)* and *Petri nets*; both allow for formal verification of the correctness of a control system. However, despite significant research advances in recent years, these formal techniques have not been widely employed in industry [9]. We believe that such approaches are still low-level formalisms, resulting in large and unwieldy systems. Furthermore, there are few understandable formalisms by industrial engineers. The *Statecharts* formalism was described by David Harel [10] in the 80s to make the specification and design of complex DES easier. It extends conventional finite state machine with notions of hierarchy, concurrency, and communication.

The use of *Statecharts* has been well explored in the industrial automation field. [2], [11], [12], [13], [14], [15] and

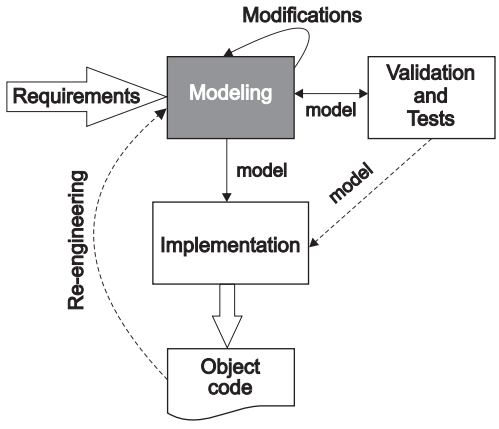

Fig. 1. Application life-cycle: overview

[16] present contributions to this field. However, based on our knowledge, all of them show high dependence on designer's technical knowledge and none of them describe a systematic method to create the *Statechart* model of the process/plant.

With the intent of reducing the gap between informal requirements and the modeling process, in this paper, we propose a high-level methodology for modeling the plant, and control of industrial applications based on components reuse and composability. The plant model is generated with use of *UML/Statecharts* and scenarios, which are defined by a sequence of events like *UML/Sequence diagram* to determine the correct operation of entire applications or part of its components. The control model is generated by the scenarios specified in previous step, also using *UML/Statecharts*.

The remainder of this paper is organized as follows: Section II discusses about the main aspects of its usage in modeling of industrial automation systems. Section III describes in general the methodology proposed by this contribution. A case study is presented in Section IV, which shows a complete example of an industrial application using our approach. In the last Section, we conclude with a discussion about some future plans.

II. UML 2.0: State Machine Diagram

The *Unified Modeling Language (UML)* was proposed by Object Management Group (OMG) [17] as the standard modeling language for software and systems development [18]. With the definition of UML, *Statecharts* were adopted to describe dynamic aspects of behavioral view of the entire system or of a particular function, including control and timing. UML describes *Statecharts* with slightly different semantic from original semantic proposed by Harel in [19]. In release (UML 2.0), *Statecharts* are called *State Machine Diagram*. However, it presents few changes with regard to older releases of *Statecharts*.

Statecharts are used to model components, which can be in different states and have these states connected by transitions. A transition represents a state change, or how to get from one state to the next one. As an example, we can cite a lamp behavior that can assume states "ON" or "OFF" when triggered by an event, such as the pressing of a button.

The use of *Statecharts* and consequently *State Machine Diagram* in event-based system presents some advantages as:

- It is graphical, intuitive, and simple formalism;
- It allows *state-levels* (i.e., clustering, unclustering, and refinements) by *OR decomposition*: if the system is in an OR-state, then it must be in one of its own substates. Clustering and unclustering let designers to explore the idea of "zoom-in" and "zoom-out" for detailing or hiding system model abstraction levels;
- It permits *state orthogonality* (i.e., independence and concurrency) by *AND decomposition*. This property means that, being in a state, they must be in all of its AND components;
- It ensures time constraints using implicit timers. Formally, this is done by *timeout(e,t)* event expression, which represents the event that occurs elapsed *t* time units from the occurrence of the specified *e* event;
- Specifications of *actions and conditions* for transition events and *activities* triggered on entry, on exit or on state (throughout) can be generated. Thus, activities are durable – they take some time – whereas actions are instantaneous. Formally, the transition event is specified by $e[c]/a$ expression, in which the event *e* must occur only if an optional condition *c* is true. In this case, the action *a* must be triggered automatically;
- It is part of Systems Modeling Language (SysML) (see [20]). SysML is a general-purpose modeling language for systems engineering applications, which supports the specification, analysis, design, verification and validation of a broad range of complex systems. These systems may include hardware, software, information, processes, personnel, and facilities.

We believe that state machine diagrams are a formalism which contemplates the demands of industrial applications specification.

III. Methodology

In industrial applications, normally the *controller software* is verified in conjunction with a model of the plant in which it operates. So, it is necessary to obtain an accurate model to maintain fidelity with the real plant - relation one-to-one.

A. The Plant

For plant modeling, our methodology is based on the *hybrid approach - bottom-up and top-down*. More specifically, it proposes to model the basic elements, grouping them into larger structures. This process is repeated until it generates the correct model of application. It is based on *UML/Statecharts* and scenarios which are defined by a sequence of events. In our context, the *Statecharts* diagrams model the internal dynamic of the components, and the sequence of events acts like a connector to control the components. The methodology consists of four phases described as follows:

1) Modeling the basic application elements or using models already defined in a component repository;
2) Decomposing the basic states in substates, if necessary;
3) Representing all automation plant components as parallel states;
4) Making scenarios based on a sequence of events to extend each state and to include special characteristics to generate the right application behavior.

Phases 1 and 2 consist of modeling and refinements of the basic elements which compose the application. They can be run several times as an iterative process. In each iteration, we work with components which are more and more complex. Further, these components can be grouped in a repository. The third phase determines that all application components must be executed at the same time, in a parallel way. The last step of the methodology (phase 4) is responsible for components interconnection. That is, it adds actions to the basic transitions that permit interaction among components

in the application. Phase 4 is understood as cards (defined by scenarios) which must be inserted in the application basic model. They allow the inclusion of additional functionalities to the basic use of application, as well as enabling the simulation and validation of the application behavior. We will present how our methodology works below:

1) Modeling of Basic Components: For automation systems, many components follow an *On/Off pattern*. Fig. 2-a shows the dynamic behavior of this pattern, which can be in states: "OFF" or "ON", and two transitions to change from state: "e1" from "OFF" to "ON" and "e2" from state "ON" to "OFF". Other components require adjustment in modeling to include new characteristics. For example: a temporary state (Wait) between the states "On" and "Off" (see Fig. 2-b).

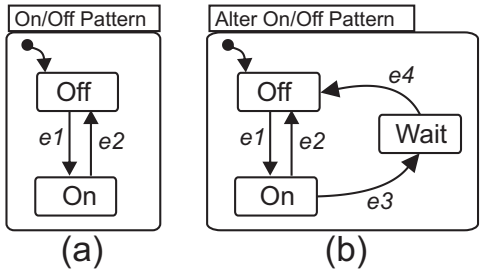

Fig. 2. On/Off pattern: basic model

In the manufacturing field, one of the most common components is the pneumatic cylinder that can be composed of more simple components (arms, valves, and sensors) and can have displacement sensors/end-position initiators.

Fig. 3 depicts a single-action cylinder with advancing controlled by the valve, return carried through springs, and one end-position sensor which is triggered when the cylinder arm gets the full advance. The generic notation "a/b" in a transition indicates that some dependence exists between the components. The component in which the action "b" is executed depends on another component with which the event "a" is triggered. So, the action "b" will be triggered only when the event "a" occurs. Therefore, as the figure shows, the transition "vOn/ac" indicates that: the cylinder arm depends on the valve, i.e., the arm advances while the valve remains open. When the event "vOff" occurs the cylinder arm gets "Returned". The cylinder arm has the following behavior: when the event "ac" occurs, the arm gets to "Advancing" in a specified time, which depends on technical characteristics and it is represented by "*" in the figure. If the valve is closed before this specified time (event *tm(t)*), the cylinder arm gets to "Returned" and nothing happes to the sensor. If the event *tm(t)* occurs, then the arm gets to "Advanced" and the active state of the sensor passes from "False" to "True", implicitly. So, when the valve is closed, the arm gets "Returned" and the sensor passes from "True" to "False".

The scenario that describes the desired operation of the cylinder is very simple: one external event "ev" allows the opening of the valve - transition "ev/vOn"; and after the sensor

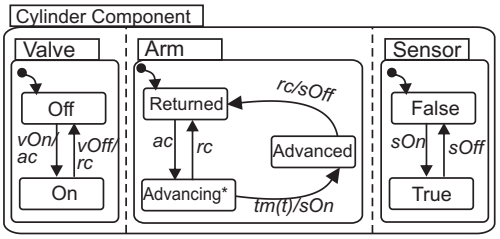

Fig. 3. Single-action cylinder: basic model

detects the total advance of the cylinder-arm, the valve must be closed - transition "sOn/vOff".

B. The Control

The control model is generated by the scenarios specified in phase 4 of our methodology as follows: for each actuator of the plant model there is a *Statechart* diagram with two states: "Off" and "On", and transitions to turn on/turn off. The transitions are specified by the *"event[condition]/action"* expression, and the sensors can be used to create such conditions. Fig. 4 shows the basic behavior of a general actuator, which follows the *On/Off pattern*. As the figure shows, the actuator G can be in states: "Off" or "On"; the transitions "ev[c1]/aOn" and "ev[c2]/aOff" change state, if the guard conditions $c1$ and $c2$ were true. In this case, the actions aOn and aOff are triggered, respectively. It is important to mention that $c1$ and $c2$ can be generated, in a systematic way, from the sequence of events (scenarios).

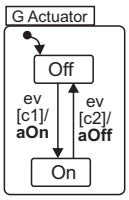

Fig. 4. Generic actuator: basic behavior

With the basic components, different models of the controlling software and different plant executions can be generated. The *control model* must provide the organization's policy for operating the actuators in the plant model. Moreover, the model validation can be analyzing some properties of *Statecharts* verified through the reachability tree of the complete model, such as: i) *reinitiability* – for each state configuration sc_i reached from the initial configuration sc_0, is it possible to come back to sc_0 by a sequence of events? ii) *dead component* – does the controller act in all of the components in the model? iii) *deadlock* – is there a state configuration sc_i in which progress cannot be made, because no transition is able to be trigged?

IV. Case Study

In this section, we describe a complete example of a typical system in the manufacturing area using the our methodology.

A. Definition

The tagged machine (see Fig. 5) is composed of seven components: (a piece-loader, a piece-sensor – PS, a feeding cylinder – C1, a pressing cylinder – C2, an extraction cylinder – C3, an air-compressed valve – V4, and a photoelectric sensor – FS). All cylinders are single-action cylinder with returning spring, and their advances are controlled by valves. Each cylinder has a displacement sensor and end-position initiators. The extraction of manufactured piece is made through the fourth air-compressed valve (V4) which is controlled by a photoelectric sensor.

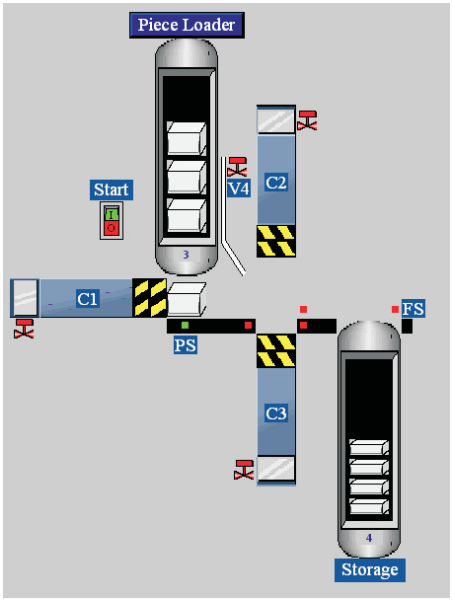

Fig. 5. Tagged machine: simulation

In its initial configuration the system needs to have all cylinder arms, valves and sensors set to "RETURNED", "OFF" and "FALSE", respectively. The system starts its operation when the "Start" button is pushed. The running scenario is described as follows:

> "The feeding cylinder pushes one workpiece to the mold. Then the pressing cylinder pushes the tag over the workpiece for about two seconds. As the last step, the extraction cylinder joins valve4 to remove the piece, pushing it to the deposit."

For controller validation, the tagged machine was simulated using the execution environment developed by *the Jakarta Project Commons SCXML* [21] and the tools *Elipse SCADA* [22]. The Jakarta Project Commons SCXML provides a generic event-driven state machine based on execution environment, borrowing the semantics defined by SCXML (State Chart XML). The Commons SCXML library can be used by frameworks needing a process control language. The Application Programming Interface (API) – SCXML 0.6 API - defines several Java classes and interfaces to control an application. Elipse SCADA is, at the same time, accessible, friendly, and totally flexible. It is a tool for automation, which eliminates the need for lengthy, expensive solutions, enabling the process with competitiveness, efficiency, and quality. Fig. 6 shows the general structure of the case study. The applications that represent the plant model and animation run in a same computer and the communication between them is got by *OPC specification* [23]. The control model run in a different computer and communicate with the plant model by the message passing by means of a java sockets, in agreement with the client/server approach.

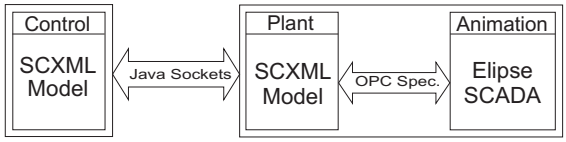

Fig. 6. Case study: general structure

B. Control and Plant Modeling

This example uses basically cylinder components which were discussed in subsection III-A, however, we will consider the existence of a repository with such components and we will describe here only the stages 3 and 4 of our methodology. So, in agreement with state 3, the modeling process represents all plant components in state *"InOperation"* of a generic application (see Fig. 7). All components must be executed at the same time, in a parallel way. This figure shows a textual description, which: the first sentence *"Cylinder1* ***is*** *SingleActionCylinder;"* indicates that $cylinder1$ is composed of three On/Off components: cylinder arm1, valve1, and sensor1; the next two sentences say that $cylinder2$ and $cylinder3$ are also composed of three On/Off components: cylinder arm, valve, and sensor with their dependencies. The dynamic of the cylinders corresponds to component shown in Fig. 3; the $valve4$ is composed of one On/Off component: valve; and, finally, $photoSensor$ and $pieceSensor$ are On/Off components: sensor. The graphical representation of the state *"InOperation"* of the model can be got directly from textual description (see Fig. 8).

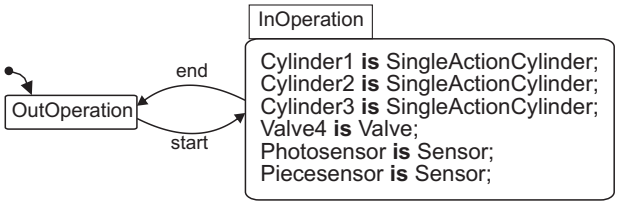

Fig. 7. Tagged machine: basic model

For each sensor in the plant model is defined one boolean variable in the data area through <*datamodel*> element, based on SCXML specification. Therefore S1, S2, S3, FS, and PS are boolean variables that represent the state of the sensor1, sensor2, sensor3, Photosensor, and Piecesensor, respectively.

From the basic plant model, scenarios can be defined to represent the right behavior of the application. Fig. 9

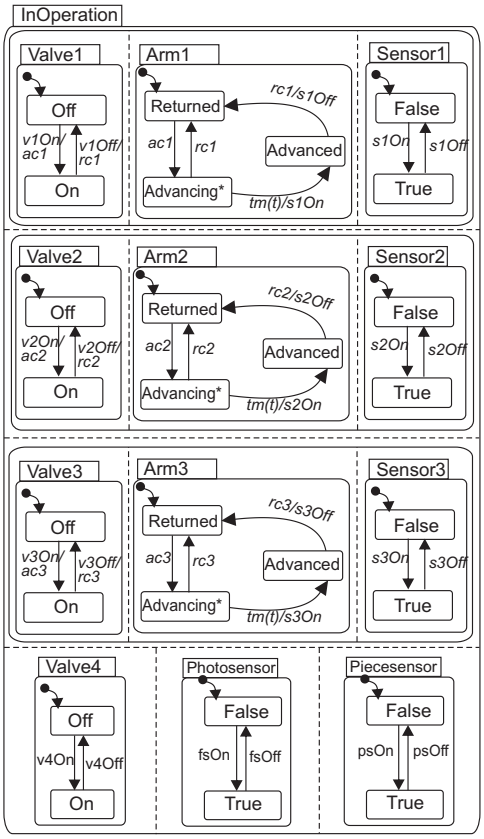

Fig. 8. Tagged machine: Statechart model

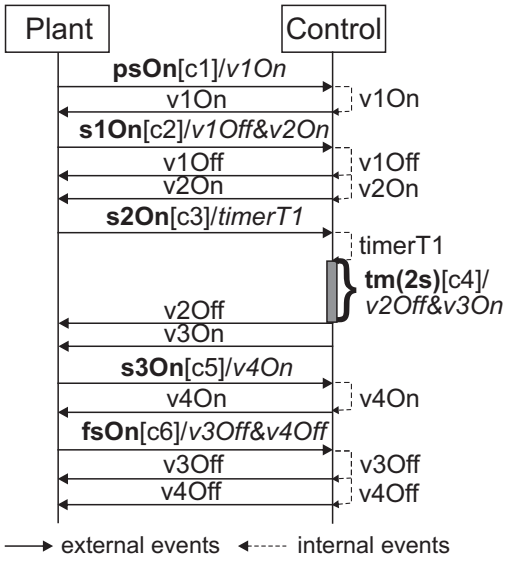

Fig. 9. Tagged machine: sequence of events

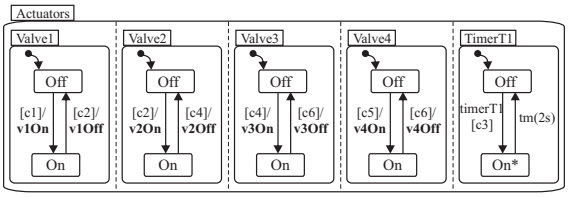

Fig. 10. Tagged machine: control model

illustrates the running scenario described in subsection IV-A, which corresponds to the stage 4 of our methodology. In agreement with this figure, when the sensors are perceived in the plant model, command-events to open and/or close valves are triggered automatically in the control model. For example, $psOn[c1]/v1On$ means that: when the event "psOn" occurs, the command-event "v1On" is triggered (i.e., when the $Piecesensor$ detects one workpiece, the $valve1$ must be opened). In this case, $[c1]$ represents a guard condition that must be truth when the event "psOn" to occur (i.e., S1, S2, S3 and FS do not detect anything, and PS detects one workpiece); $s1On[c2]/v1Off\&v2On$ means that: when the event "s1On" occurs, if $[c2]$ is true then the command-events "v1Off" and "v2On" are triggered (i.e., when the S1 detects total advance of cylinder-arm1, the $valve1$ must be closed, and the $valve2$ must be opened). In this case, $[c2]$ consists in S1 detects total advance of cylinder-arm1, and S2, S3, FS and PS do not detect anything; $s2On[c3]/timerT1$ means that: when the event "s2On" occurs, if $[c3]$ is true then the event "timerT1" is triggered in the internal component $timerT1$, which should be included in the control model. There should also be a boolean variable T1 associated with this timer, in the <*datamodel*> element. In this transition, $[c3]$ means S2 detects total advance of cylinder-arm2, and S1, S3, FS and PS do not detect anything; When the timer $timerT1$ gets timeout (in this case, two seconds), it must change variable T1 from value "false" to "true", and trigger the command-events "v2Off" and "v3On" if the $[c4]$ is true. In this case, $[c4]$ means S2 detects total advance of cylinder-arm2, and S1, S3, FS and PS do not detect anything, and T1 equals "true"; and so on. It is important to mention that our animation generates other events (e.g., "psOff", "s1Off", "s2Off", "s3Off", and "fsOff"). However these events change only the value of associated variables and colors in the simulation.

In the control modeling, for each actuator of the plant model is defined a Statechart diagram with transitions to turn on/turn off. The variables of the <*datamodel*> element are used to specify guard conditions in the transitions. Fig. 10 shows the control model generated in agreement with the scenario presented in Fig. 9. In this figure, the asterisk in state "On" of TimerT1 indicates the start of timer and after two seconds (tm(2s)) the variable "T1" should be changed to "true".

C. The Controller

After analysing the model properties, the *Ladder diagram* was carried out by the control model shown in Fig. 10. Each transition of the control model results in a "rung" of the *ladder*, which the guard condition represents boolean expressions based on sensors and/or temporary variables to set/reset the actuators and/or timers. For example: the transition "[c1]/v1On" results in the line 0 of the ladder diagram

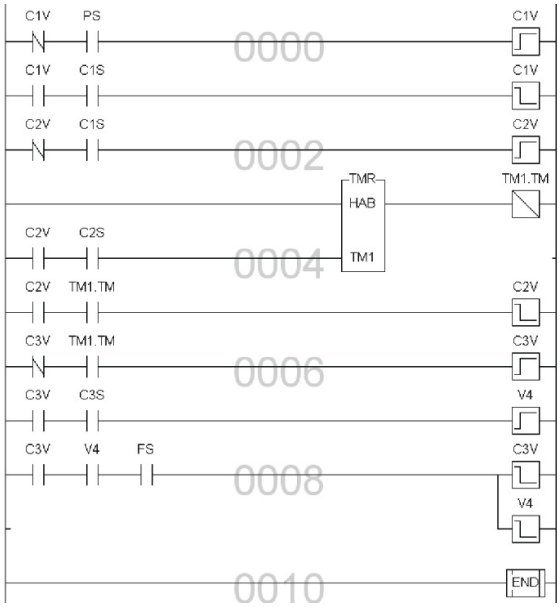

Fig. 11. Tagged machine: Ladder diagram

(see Fig. 11), which the condition [c1] must be truth, and the valve1 must be opened (i.e., the output associated with this actuator must be setted); "[c2]/v1Off" results in the line 1, which the condition [c2] must be truth, and the valve1 must be closed (i.e., the output associated with this actuator must be resetted); [c2] also represents the condition of transition "[c2]/v2On", which indicates that the valve2 must be opened. Thus, the line 2 is generated. The valve2 is closed when the transition "[c4]/v2Off" occurs; and so on. Fig. 11 shows the complete *ladder diagram* resulting of this process.

The name of the events and states were renamed in the Ladder code, due syntax of the PLC used, for example, the command-events $ViOn$ and $ViOff, (i = 1, 2, 3)$ were renamed to $CiV, (i = 1, 2, 3)$ and the sensors $Si, (i = 1, 2, 3)$ were renamed to $CiS, (i = 1, 2, 3)$. Moreover, some optimizations in the transitions were realized, considering that the system starts with all cylinder arms in the position returned and all sensors without detect workpiece. In the translation of each transition, the state source was included in the *"rung" Ladder* as basic condition. Thus, the guard conditions became simpler in the final code.

V. Conclusion

In this paper, we presented a methodology for systematizing the process of plant and control modeling of manufacturing systems. Our proposal uses the *UML/Statecharts* associated with scenarios based on a *sequence of events*. It has four phases which can be executed as many times as necessary. Altogether, this methodology represents a *hybrid approach - bottom-up and top-down*, allowing components reuse and keeping a one-to-one relation between plant and model (i.e., it is faithful to the actual system). From the scenarios, we generated the control model also using *UML/Statecharts*. The control reduces the state space of the plant model, and makes only the desired operation possible. One typical example of the manufacturing application was described in detail to illustrate our proposal.

Currently, we are developing a prototype, in Java language, for the creation and simulation of models generated by our methodology. In parallel, we intend to formalize some properties on the *UML/Statecharts*, besides the adapt an algorithm for creating a reachability tree. Finally, we believe that some component patterns could be specialized and, consequently, some connectors (such as **is, with, contains**) can be used to formalize a textual language for our research.

References

[1] IEC, "International eletrotechnical commission. programmable controllers part 3, programming languages," IEC61131-3, 1993.

[2] M. Bani Younis and G. Frey, "UML-based approach for the reengineering of PLC programs," in *32nd Annual Conference of the IEEE Industrial Electronics Society (IECON'06)*, 2006, pp. 3691–3696.

[3] B. Boehm, "A view of 20th and 21st century software engineering," in *ICSE '06: Proceeding of the 28th international conference on Software engineering*. New York, NY, USA: ACM Press, 2006, pp. 12–29.

[4] IEC, "International eletrotechnical commission. functions blocks part 1, architecture," IEC61499-1, Geneva: IEC, 2005.

[5] V-Modell XT, "Part 1: Fundamentals of the v-modell," http://www.v-modell-xt.de/, 2004.

[6] W. W. Royce, "Managing the development of large software systems," in *Proc. of IEEE WESCON*, 1970, pp. 1–9.

[7] B. W. Boehm, "A spiral model of software development and enhancement," pp. 61–72, May 1988.

[8] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, 1999.

[9] E. Endsley, E. Almeida, and D. Tilbury, "Modular finite state machines: Development and application to reconfigurable manufacturing cell controller generation," *Control Engineering Practice*, vol. 14, no. 10, pp. 1127–1142, 2006.

[10] D. Harel, "Statecharts: A visual formalism for complex systems," *Science of Computer Programming*, vol. 8, no. 3, pp. 231–274, 1987.

[11] I. Bates, E. Chester, and D. Kinniment, "A case study in the automatic programming of a PLC based control system using statemate statecharts," *UKACC International Conference on Control. IEEE*, pp. 832–837, 1998.

[12] J. Machado, F. Louni, J. Faure, J. Lesage, J. da Silva, and J. Roussel, "Modelling and implementing the control of automated production systems using statecharts and PLC programming languages," in *European Control Conference (ECC2001)*, Porto, Portugal, 2001.

[13] Y. Huang, M. Jeng, and C. Hsu, "Modeling a discrete event system using statecharts," *International Conference on Networking, Sensing & Control. IEEE*, vol. 2, pp. 1093–1098, 2004.

[14] K. Sacha, "Automatic code generation for PLC controllers." in *SAFECOMP*, 2005, pp. 303–316.

[15] Z. Hu and S. M. Shatz, "Explicit modeling of semantics associated with composite states in UML statecharts," *Automated Software Engg.*, vol. 13, no. 4, pp. 423–467, 2006.

[16] C. Secchi, M. Bonfe, and C. Fantuzzi, "On the use of UML for modeling mechatronic systems," in *IEEE Transactions on Automation Science and Engineering*, vol. 1-4, 2007, pp. 105–113.

[17] OMG, "Object management group. UML superstructure. part ii - behavior. chapter 15 - state machines," http://www.omg.org/docs/formal/05-07-04.pdf, 2005.

[18] K. Hamilton and R. Miles, *Learning UML 2.0*. O'Reilly, 2006.

[19] D. Harel and M. Politi, *Modeling Reactive Systems with Statecharts: the Statemate Approach*. McGraw Hill, 1998.

[20] SysML, "Systems modeling language," http://www.sysml.org/, 2006.

[21] SCXML, "The jakarta project commons SCXML," http://jakarta.apache.org/commons/scxml/, 2006.

[22] Elipse Software, "E3 and elipse SCADA products home," http://www.elipse.com.br/elipse/produto_apresentacao.aspx?id=2, 2007.

[23] OPC Foundation, "The OPC foundation – dedicated to interoperability in automation," http://www.opcfoundation.org/, 2007.