Simulation of Industrial Applications using the Execution Environment SCXML

Raimundo Santos Moura and Luiz Affonso Guedes

Abstract—The design engineers need to use high level formalisms to facilitate the development, maintenance, and documentation of the industrial control systems. These formalisms must permit the modeling, formal validation, and code generation to PLC. This paper discusses statechart for modeling of industrial systems, and presents a case study that simulates the general schema of the industrial automation process, based in PLC. The case study was implemented with java technologies, and was simulated using the execution environment developed by the Jakarta Project Commons SCXML.

I. INTRODUCTION

The experienced evolution in the industrial automation field has been always associated with the technologies developed and supported by several suppliers. So, in a few decades, we have passed by pneumatic technology, analog electronic technology, and digital electronic technology. In the current days, we are using a technology tightly based in software that establishes a new branch in automation, called industrial informatics. The industrial informatics perceives the activities that compose the industrial automation process as an integrated and flexible system. Therefore, we need tools that will help the development, tests, and reconfiguration of systems based in software in an efficient and safe way.

Industrial automation uses the systems theory to control machines and processes. Many attempts have been made to develop notations and semantics that are used to classify and describe the different kinds of systems in modeling area. Such attempts provide an infrastructure needed to resolve some actual problems of engineering, and aim to increase the productivity, quality, and safety of the system development process.

In the computer science, several models guide the software development process such as the V-Model [1], that was developed in Germany to regulate this process and became a classical model to the planning and execution of projects; the Waterfall Model [2], that is a sequential software development model in which development is seen as sequence of phases; and the Spiral model [3], that is an iterative software development model which combines elements of the stages of software design and prototype.

In short, the application life-cycle can be divided in three phases: Modeling - Validation - Implementation (see figure 1). The “Modifications” arc represents multiple iterations that can occur in software modeling processes. In each phase, some properties are more important than another, and they determine advantages and disadvantages between the formalisms.

R. Moura is with the Department of Informatics and Statistics / UFPI; (e-mail: rmoura@dca.ufrn.br)
L. Affonso is with the Department of Computer Engineering and Automation / UFRN; (e-mail: affonso@dca.ufrn.br)

It is important to mention that dynamic systems simulation has been used as one of the approaches to the validation phase.

Fig. 1. Application life-cycle: overview

In the modeling phase, the Unified Modeling Language (UML) is recommended specification by the Object Management Group (OMG). Jon Siegel, in [4], says that “the OMG’s UML helps you specify, visualize, and document models of software systems, including their structure and design, in a way that meets the requirements for scalability, robustness, security, extendibility, and other characteristics”.

Figure 2 shows an overview of industrial automation process, based in Programmable Logic Controllers (PLC). The sensors are installed in plant module and generate events that represent the PLC input variables. The actuators are associated with the actions produced by PLC program and represent output variables.

Fig. 2. Overview of industrial automation process based in PLC

The IEC-61311-3 standard [5] defines five programming languages for PLC: ladder diagrams, function block diagrams, sequential function charts, structured text, and instruction list. However, all these languages are considered low level, and would be interesting to designers to use a high level formalism to facilitate the development, maintenance, and documentation of the industrial control systems.
In agreement with these needs, the main aim of this paper is to explore the use of statechart formalism [6] to modeling of industrial systems, and to make simulations with the execution environment developed by the Jakarta project Commons SCXML [7]. The UML 2.0 defines the statecharts with the name state machine diagrams. Nevertheless, they maintain the same semantic of the Harel statecharts. Therefore, the term statechart will be used along this work.

The remainder of this paper is organized as follows: section II presents a short overview of statechart formalism, and discuss the main aspects of its usage in modeling of industrial automation systems, based in PLC. Section III describes some informations about the Jakarta Project Commons SCXML. In section IV, a case study is presented showing the animation of items tagged machine. Finally, in section V we present the conclusions and some future works.

II. STATECHARTS

The Statechart formalism was described by David Harel [6] in the 80s to make easier the specification and design of complex discrete-event systems [8]. It extends conventional state-transition diagrams with the notions of hierarchy, concurrency, and communication. Statecharts constitute a visual formalism for describing states and transitions in a modular way, enabling clustering (abstraction), orthogonality (concurrency), successive refinements, and encouraging “zoom” capabilities for moving easily back and forth between levels of abstraction.

In our opinion, the main advantages to use this formalism in modeling event-driven systems are presented below:

- It is graphical, intuitive, simple, and complete formalism;
- it supports resolution of the state-explosion problem of the finite state machines, when the complexity of the system increases;
- It permits 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 own substates. Clustering and unclustering permit designers to explore the idea of “zoom-in” and “zoom-out” for hide or detail abstraction levels of the system model;
- It permits the 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 using the event expression timeout(event, number), which represents the event that occurs precisely when the specified number of time units have elapsed from the occurrence of the specified event;
- It permits to specify actions and conditions for transition events, and activities which can be triggered on entry, on exit or on state (throughout). Thus, activities are durable - they take some time - whereas actions are instantaneous. Formally, the transition event is specified by expression e[c]/a, 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 Unified Modeling Language (UML), which is a widespread language in industry and academia.

Therefore, there is a guarantee of continuing qualification of human resources.
- It is also part of Systems Modeling Language (SysML) [9]. SysML is a domain-specific modeling language for systems engineering applications, which supports the specification, analysis, design, verification and validation of a broad range of systems and systems-of-systems. These systems may include hardware, software, information, processes, personnel, and facilities.

We believe that statecharts are a powerful formalism which contemplate the needs of industrial applications specifications. Furthermore, they can be simulated by the Jakarta Project Commons SCXML that will be described in next section.

The complete description of statecharts, that was applied to famous clock Citizen Quartz Multi-Alarm III, can be found in [6]. There is also the book “Modeling Reactive Systems with Statecharts” [10], by David Harel and Michal Politi, which provides a detailed description of a comprehensive set of languages for modeling reactive systems. The UML/Statecharts semantics recommended by Object Management Group (OMG) are available in [11].

III. THE PROJECT COMMONS SCXML

The World Wide Web Consortium (W3C) published recently the State Chart XML (SCXML) recommendation [12] which defines a complete specification to represent the statecharts diagrams by a XML file. SCXML combines concepts from CCXML and Statecharts (Harel State Tables). CCXML [13] is also a W3C recommendation, which was published in year 2005. It is an event-based state machine language designed to support call control features in Voice Applications (specifically including VoiceXML, but not limited to it). The CCXML 1.0 specification defines both a state machine and event handing syntax and a standardized set of call control elements.

In agreement with the Jakarta project official site, Commons SCXML provides a generic event-driven state machine based in execution environment, borrowing the semantics defined by SCXML. Most things that can be represented as a UML statechart - business process flows, view navigation bits, and interaction or dialog management - can leverage the Commons SCXML library. The library can also be used by frameworks needing a process control language.

The Application Programming Interface (API) – SCXML 0.5 API defines several Java classes and interfaces to control an application. The main classes of SCXML 0.5 API are:

- public abstract class AbstractStateMachine – demonstrates one approach for providing the base functionality needed by classes representing stateful entities, whose behaviors are defined via SCXML documents. It receives a SCXML document that describes the “life-cycle” of the instances of the application, and has some methods that permit fire an event on the SCXML engine, get the SCXML object representing this state machine, get the SCXML engine driving the “life-cycle” of the instances of this class, and reset the state machine;
- public class SCXMLExecutor – represents the SCXML “engine” that executes SCXML documents. The particular semantics used by this engine for executing the
SCXML are encapsulated in the SCXMLSemantics implementation that it uses. There is a default implementation that presents the basic functionality and permits to get the current status, the root context for the execution, and the state machine that is being executed. In case study, described in section IV, no modification was made in this class, however, we use some methods to get information about the engine current status:

- **public class SCXML** – corresponds to the `<scxml>` rootelement, and serves as the “document root”. It has methods for add a child state, get the children states, add a target to this SCXML document, get the targets map, which is a map of all basic states and parallel states associated with this state machine keyed by their id, and set or get the ID of the initial state;

- **public interface SCXMLListener** – consists in the “Listener” interface for observable entities in the SCXML model. Observable entities include SCXML instances (subscribe to all entry, exit and transition notifications), State instances (subscribe to particular entry and exit notifications) and Transition instances (subscribe to particular transitions). It has three methods onEntry, onExit and onTransition to handle the entry into, the exit out, and the transition, respectively. In our case study, onEntry methods were used to simulate time constraints.

The project Commons SCXML has been used for choreographing the execution of speech components taking part in a voice interaction bounded by a Reusable Dialog Components (RDC) group container, and for managing the flow of control in web based applications, using shale dialogs. We are using SCXML for modeling and simulation in the industrial automation area.

Current information about the project SCXML, including documentation, releases, and resources can be access in [7].

IV. CASE STUDY

This section presents a case study that realizes a simulation of the tagged machine. The case study was developed in agreement with basic features that are showed in figure 3. The **control** is represented by java class AutomationSC.class, and consists in the SCXML engine. The **process** is represented by Java class Automation.class, and consists in the Java 2D animation. The control and process can be run in different computer systems and the communication between them can be made through message passing, by means of Java sockets.

A. The Plant

The tagged machine (see figure 4) is a composite of six elements (a pieces loader – not showed in the simulation, a feeding cylinder – cylinder1, a pressing cylinder – cylinder2, an extraction cylinder – cylinder3, a valve air-compressed – valve4, and a photoelectric sensor – sensor). All cylinders are single-action cylinder with return spring, and their advances are controlled by valves. Each cylinder has a displacement sensor and end-position initiators. The extraction of work piece is made by the fourth valve air-compressed (valve4) that is controlled by photoelectric sensor.

![Tagged Machine: Plant Simulator](image)

**Fig. 4.** Case study: tagged machine

The initial configuration of system needs that all cylinders have been “returned”, the valves have been “off”, and the sensor has been “false”, i.e., it doesn’t detect the presence of piece. The system starts the operation by pressing the “start” button and it is turned off, automatically, when the photoelectric sensor detects the presence of work piece. The scenario of running is described bellow:

“The feeding cylinder pushes one work piece until the mold; the pressing cylinder presses the tag over the piece during two seconds; the extraction cylinder joins valve4 to make the piece removing, pushing it for the deposit.”

B. The Model

The modeling process was realized by the use of statecharts formalism and it consists in the application of four phases that are described follow:

1) To model the basic elements of application (for example: cylinders, valves and sensors) or to use the models defined by library collections;

2) To decompose the basic states in substates such as a cylinder that can be “RETURNED” or “ADVANCED”,

![Case study: general structure](image)

**Fig. 3.** Case study: general structure
a valve that can be “OFF” or “ON”, a sensor that can be “FALSE” or “TRUE”, etc;

3) To represent all components of automation plant as parallel states that inherit characteristics of base elements, if possible;

4) To make scenarios based in events sequence to extend each state and to include special characteristics to generate the correct behavior of the application.

Figure 5 shows the events sequence that is trigged to scenario presented in this case study. The dashes lines represent internal events and describe special characteristics to generate the correct behavior of the application. The solid lines represent external event which are sensed by the plant.

The final model consists in the parallel running of three cylinders, four valves, three end-position sensors of the cylinders and one photoelectric sensor (see figure 6).

---

**C. The Implementation**

This case study was implemented using a Java platform and the SCXML 0.5 API. We create two classes that run in different computer systems and communication between them is got by message passing by means of java sockets, in agreement with the client/server approach.

---

**D. The Simulation**

Internal events such as open/close a valve or advance/return a cylinder are triggered to the plant, and are simulated by graphical animation. At the end of an animation (for example: the total advance of cylinder), the plant sends an external event that represents the end-position sensor of this cylinder. These events generate a new configuration of current states.
Fig. 7. Segment of SCXML code that defines the cylinder1

To simulate the complete animation of the running cycle of tagged machine, the external event “start” is triggered by the pressing of the Start button in graphical application. The other external events that appear in figure 5 are sent automatically always in the end of animation of event and immediately before its occurrence. For example, the event “s1” (i.e., the end-position sensor of cylinder1) is sent automatically after the animation of internal event “v1” (i.e., after the valve1 gets opened); the event “s1Off” (i.e., the end-position sensor of cylinder1 is false) is sent after the animation of internal event “v1Off” (i.e., after the valve1 gets closed); the event “s2” (i.e., the end-position sensor of cylinder2) is sent automatically after the animation of internal event “v2” (i.e., after the valve2 gets opened); and so on.

In the low part of figure 5, the event “fs” (i.e., the photoelectric sensor is detecting presence of the work piece) is sent after the internal event “v4Off” (i.e., after the valve4 gets closed), and then the self sensor sends automatically the event “fsOff” (i.e., the photoelectric sensor isn’t detecting presence of the work piece), which sends the event “stop” to finalize the running cycle of the tagged machine, and to enable again the start button in animation. This cycle is repeatedly run whenever the start button is pressed.

A simple change of scenario applied in the modeling process (cf. phase four of process described in subsection IV-B) produces a new behavior in the automation process. That discussion is out of the scope of this paper, but we will explore it minutely in future works.

E. Project Commons SCXML: Limitations

In the implementation of this case study, we perceive some limitations of the Jakarta project Commons SCXML, such as:

- For each state of SCXML model produced, we have to add a method in Java application that extends the class AbstractStateMachine, although the method doesn’t run any specific action;
- Time events, like: “the pressing cylinder presses the tag over the piece by two seconds”, that is represented in SCXML by tag <send delay="2s" event="V2Timer"/>, don’t generate the correct behavior of the application. In current release of SCXML engine (version 0.5), the attribute delay isn’t considered. The SCXML engine simply runs without waiting for the specified time. Therefore, we have made a small adaptation to generate the correct behavior of the application.
- Parallel states are presented in each configuration of engine in random order. This makes more difficult the understanding of the log file produced by running of application.

V. CONCLUSION

This paper presented a case study on the use of SCXML engine, which was developed by the Jakarta project Commons SCXML, for simulation of the industrial automation process. We implement two Java classes to simulate the plant and to model the control of the application. The classes run in different computer systems and the communication between them can be made through message passing by means of java sockets.

The use of Java SCXML engine was very encouraging, despite of some limitations already presented in this work. Therefore, we intent to explore the use of SCXML engine in function of advantages offered by the Java technology.

Currently, our researches are directed to the development of the methodology to act in all life-cycle of the industrial systems, i.e., in modeling, validation, and implementation phases. Future works will include: i) to describe in detail the modeling process that was used in the case study; ii) to formalize some properties on the UML/Statecharts produced; and iii) to generate PLC code, in Ladder diagram or in some script language for use in actual applications.

REFERENCES


Raimundo S. Moura received a B.Sc. degree in Computer Science from the Federal University of Piauí (UFPI), Brazil, in 1994 and a M.Sc. degree in Computer Science from the Federal University of Pernambuco, Brazil, in 1997. He is a professor in the Department of Informatics and Statistics at UFPI, Brazil. Currently, he is a PhD student in Electrical Engineering at Federal University of Rio Grande do Norte, Brazil. His research interests include industrial informatics and high level modeling languages.

Luiz Affonso Guedes has a graduate in electrical engineering from Federal University of Para, Brazil, in 1987, and PhD degree in Computer Engineering from State University of Campinas, Brazil, in 1999. Since 2003, he is an associate professor in the Department of Computer Engineering and Automation at Federal University of Rio Grande do Norte, Brazil. His main research interests include Communication System to Industrial Application and High Level Programming Languages to Industrial Automation.