Int J Adv Manuf Technol (2001) 17:928–938

The International Journal of Advanced Manufacturing Technology

A Hybrid Specification Method for the Design of a Workcell Controller in Manufacturing Systems

H.-F. Lai and C.-E. Lee

Department of Industrial Engineering and Management, National Chiao Tung University, 1001, Ta-Hsueh Road, Hsin-Chu, Taiwan

This paper presents a hybrid specification method (HSM) for the description and design of a workcell controller in a manufacturing system. This hybrid method integrates the top-down IDEF0 and bottom-up statechart approaches. The high-level functional abstraction of a target system is first decomposed using IDEF0. Statecharts of the overall system are then combined using compositional constructs. Programmable logic controller (PLCs) programs that represent the low-level abstraction are finally transformed from the statecharts. To demonstrate the viability of this method, an example of a workcell controller is designed using this method. The proposed method provides an alternative to the traditional top-down decomposition technique with reusable and extendable properties.

Keywords: Behavioral specification; Functional specification; HSM approach; IDEF0; Statecharts; Workcell controller

1. Introduction

A workcell is usually a basic unit of a manufacturing system. An efficient design method for workcells is the key to achieving the goal of automation [1]. Computer tools are widely used in the following two domains for workcell automation. One domain concentrates on CAD/CAPP/CAM. The other one focuses on equipment control, including workcell controller design, shop floor control (SFC), and the manufacturing execution system (MES) [2,3]. An efficient design method for a workcell controller is still a fundamental but challenging topic, which includes the analysis and synthesis of various models including information process flow, material flow, control flow [4], and cost flow [5].

Several methods for designing a workcell controller have been proposed [1,6,7]. However, there is no single tool or language that satisfies the overall desired features of a workcell system [8]. These design methods generally addressed the linkages between various modelling tools and introduced transformation rules and implementing procedures to guide the analysis of the functional requirements and behavioural specifications of a target system. Three categories of these methodologies had been studied by a number of researchers, top-down, bottom-up, and hybrid paradigms.

For the top-down decomposition method, Wang et al. [9] proposed a multiview modelling approach, IDEM, which describes a target system from the viewpoints of function, information, and dynamics. Liang and Hong [7] proposed a hierarchical transformation method (HTM), IDEF0/CPN/G2, which analysed a manufacturing system using IDEF0 [10], simulated it using a coloured Petri net (CPN), and implemented the target system in a real-time expert system, G2. Jafari and Boucher [6] also analysed a system using IDEF0 and simulated it using CPN, but implemented it in a programmable logic controller (PLC). Uzam and Jones [11] proposed an extended Petri net method to analyse a target system and then implemented it in a PLC.

The bottom-up composition methods model the submodules in detail first, without considering the interactions among them. The submodules are then combined by taking their interactions into consideration [12,13]. The advantage of the bottom-up methods is that they can easily identify the submodules since they have physical correspondences. For instance, a workcell may consist of machines, robots, and buffers. Each component is a submodule in this workcell. However, combining components will increase the complexity of interactions among the submodules. These compositional issues will be discussed in this paper.

This paper presents a hybrid specification method (HSM), which integrates the top-down functional analysis using IDEF0 and bottom-up behavioural specifications using statecharts. The results are then translated into the PLC programs directly using the transformation rules. This method aims at state changes that are consistent in nature and are easy to understand and analyse [14,15], unlike the top-down decomposition approach, which is suitable only for stable environments. Using a bottom-up compositional strategy, the HSM meets the requirements for changes in variable environments.

Correspondence and offprint requests to: Dr C.-E. Lee, Institute of Industrial Engineering and Management, National Chiao Tung University, 1001 Ta-Hsueh Road, Hsin-Chu, Taiwan. E-mail: cclee@advtek.com.tw

The remainder of this paper is organised as follows. In Section 2, the proposed method HSM is described. Section 3 presents a workcell example to demonstrate the proposed method. In Section 4, we analyse the static structure and the dynamic behaviour of the statecharts of the example. Related work and other models are compared in Section 5. We conclude in Section 6 with a summary of what HSM can support for a workcell controller design.

2. HSM

In the development process of a system, the requirement specification is an important stage [16]. This section demonstrates how to acquire the functional specifications using IDEF0. Statecharts are then adopted to describe the behavioural specifications that lie between IDEF0 high-level functional abstractions and PLC low-level physical implementation. Finally, the HSM development procedures and the transformation rules from statecharts to PLC programs are discussed.

2.1 Functional Specifications and IDEF0

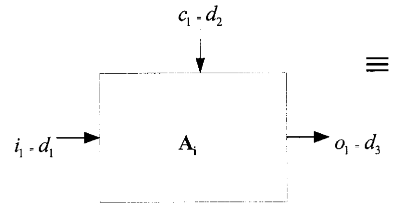
The top-down method decomposes a module into a set of submodule functional specifications. The formal top-down method applies various reduction mechanisms to decompose specifications. Functional specifications about a manufacturing system are specified in the earliest stage as a guideline for all design activities. Several tools are available for describing the functional specifications of manufacturing systems. A notable tool among them is IDEF0 (**I**CAM **DEF**inition **0**), which helps designers to capture the functional abstractions of a system. An IDEF0 functional model is an activity flow diagram that describes the computations to be performed by the system [6,7].

The functional specifications describe what functions a system has to fulfil. The IDEF0 model describes the input/output transformational relations that exist in the system. However, IDEF0 activity relationships are represented only at the graphical level with no functional foundation to interpret them. In this section, these functional semantics are interpreted using Lambda-calculus [17]. The detailed correspondence between Lambda-calculus and manufacturing systems is discussed in Liang [18].

Lambda-calculus was designed to capture the most general views in which functions or operators can be combined to form other operators. An activity box in IDEF0 can be taken as a lambda term, which is repeatedly generated from abstractions and applications. Lambda terms are defined inductively as follows [17]:

1. All constants and variables are lambda terms (atoms).
2. If x is any variable and M is any lambda term, then $(\lambda x.M)$ is also a lambda term (abstraction).
3. If M, N are lambda terms, then (MM) is also a lambda term (application).

Activity boxes are the IDEF0 building blocks, whereas arrows are flows. In the functional view, three labels ("mechanism" is omitted) are involved: inputs, controls, and outputs. Outputs are produced under the inputs, the controls being satisfied. The activity box describes a specific activity or function in a manufacturing system shown in Fig. 1. From the control logic viewpoint, the activity box indicates that if all input conditions of function A_i are satisfied, then the function A_i is evaluated, i.e. an activity A_i is executed in a manufacturing system, and the output label $o_1 = d_3$ is set to be true. These operational semantics can be expressed using β-reduction [17] as indicated in the righthand side of Fig. 1.



$$\begin{aligned} \equiv\ & (\lambda\ i_1 c_1 \cdot i_1(c_1 A_i F)F)d_1 d_2 \\ & \rightarrow (\lambda\ i_1 c_1 \cdot i_1(c_1 A_i F)F)T \\ & \rightarrow (c_1 A_i F)T \\ & \rightarrow T A_i F \\ & \rightarrow A_i, \mathit{ouput}\ \mathrm{o}_i = d_3\ \text{is}\,\mathrm{T} \end{aligned}$$

Fig. 1. The correspondence between an IDEF0 box and lambda terms.

The IDEF0 model is suitable for interpreting the high-level abstractions of a target system. However, these specifications cannot fully capture low-level interactions and dynamic information. They are not suitable for describing behavioural specifications. To alleviate these problems, statecharts are adopted in this study and are introduced in the next subsection.

2.2 State Machine and Statecharts

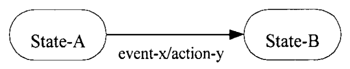
Statecharts, proposed by Harel [19,20], use a visual specification language for specifying a discrete event system. It extends the concept of finite state machines and can be described as:

Statecharts = finite state machine + depth + orthogonality + broadcast communication

Finite state machines have been widely applied as a tool for specifying the requirements of various systems [14]. A finite state machine consists of states connected by transitions as displayed in Fig. 2, where state B is the next state of A. The rounded rectangle in a state machine denotes the state. Arcs that connect the states symbolise the flow of transitions. A transition may be labelled as the form of a trigger event and an output action. This hypothetical machine can be adapted to describe the input/output relationships and state changes in the system, i.e.

$$Q_{n+1} = \delta(Q_n, I)$$
$$\text{Out} = \gamma(Q_n, I)$$

Where Q_{n+1} is the next state, Q_n is the current state, I is the input, δ is the state change function, and γ is the output function.



$$\text{next state B} = \delta(\mathrm{A}, \mathrm{x})$$
$$\mathit{output}\ \text{action}\ y = \gamma(A, x)$$

Fig. 2. Transition defined by state change.

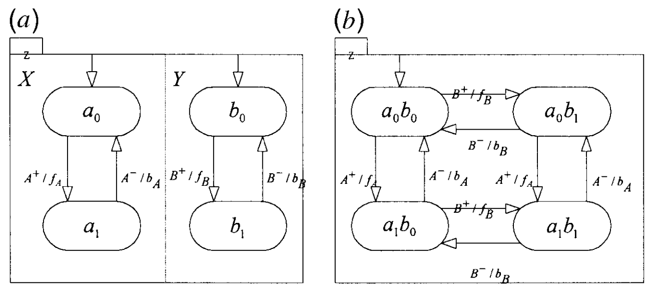


Fig. 3. The orthogonal property of statecharts.

Depth represents the different levels of detail using statecharts, i.e. states may be grouped into a superstate. For instance, an AND-state z is the superstate of the OR-states Y and Z in Fig. 3(a).

Orthogonality supports the concurrent qualities that can reduce the number of states in the graphical representation. In Fig. 3, the dashed line denotes that the two statecharts are orthogonal. Figure 3(a) presents four events. Figure 3(b) presents eight events. As the orthogonal number increases, the complexity increases exponentially [19].

Broadcast communication is the method of communication from one state to another. For instance, as an event A^+ occurs in Fig. 3, the initial state a_0b_0 will be changed into the state a_1b_0, and an action f_A will be broadcast throughout the system.

Statecharts have several admirable properties that can be applied in various application domains, e.g. manufacturing system modelling, real-time system design, and object-oriented information system development [14]. However, there are still some restrictions in the original statechart. Many extensions have been proposed to improve the descriptive ability or meet the individual requirements, e.g. objectchart [21], hierarchical state machine [22], modechart [23], extending statecharts [24], etc.

2.3 The Compositional Technique of HSM

The benefit of the compositional strategy is that a larger statechart may be composed from simpler components. This means that the existing software components can be reused. Moreover, a compositional strategy can reduce system complexity and increase system design flexibility [25]. Figure 4 depicts the general development flow of the HSM compositional strategy.

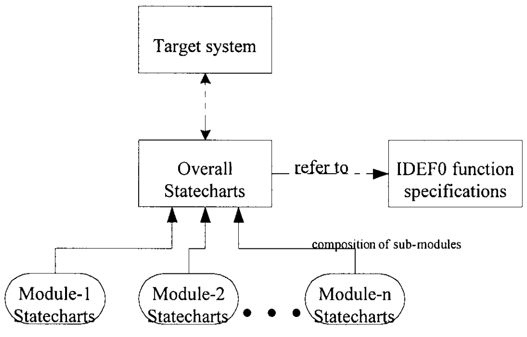


Fig. 4. The HSM compositional strategy.

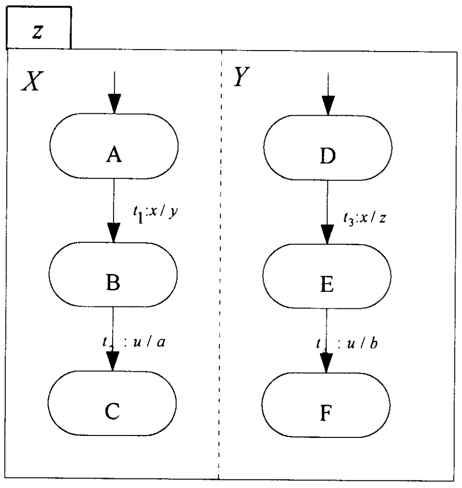


Fig. 5. Serial and parallel composition.

There are four types of compositional relationship identified from the transition interactions in this paper.

1. *Serial composition*. The output state of one transition is the input state of another transition. As indicated in Fig. 5, state X is an OR-state, where the output of transition t_1 and the input state of transition t_2 are identical. Thus, t_1 and t_2 are in serial composition.
2. *Parallel composition*. Transitions between different modules having the same input event denote a parallel composition. As shown in Fig. 5, t_1 and t_3 are parallel transitions having the same input event x, and state z is an AND-state.
3. *Merging composition*. States between different modules have the same input and output events. As indicated in Fig. 5, state B and state E have identical input and output events $\{x, u\}$, and states $\{B, E\}$ can be merged into a new common state.
4. *Union composition*. The definition domain is shared by many transitions with the resource contradiction. As shown in Fig. 6, t_1, t_2, and t_3 are contradiction transitions.

2.4 Design Steps of the HSM and Transformation Rules

PLCs have played a significant role in modern manufacturing systems. To write PLC programs for a simple system is trivial. However, as systems become more complicated, it is difficult to write or to debug the PLC programs. Grafcet [26] is a structured tool to aid the design of sequential control systems.

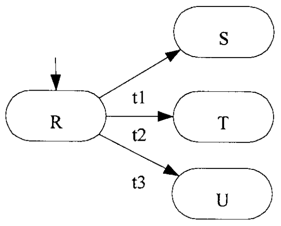


Fig. 6. Union composition.

Grafcet can be successfully applied to model simple systems with only one token in one step, but it is not as flexible as a general tool. Therefore, industrial users still prefer heuristic methods to write PLC programs [27]. In this section, a heuristic method HSM based on the statechart will be introduced.

2.4.1 Design Steps of HSM

With proper statechart syntax, the consistency and completeness of the designed system can be ensured [15]. An overall statechart model can be constructed through submodules using serial composition, union composition, parallel composition, and merging composition. The construction steps for HSM are as follows:

1. Depict the functional hierarchy of a target system:

   IDEF0 is applied to represent the overall system functional structure.
2. Depict the statechart of every submodule of the system according to the activity flow diagram of each element in step 1. This step consists of three tasks, analysing the acceptable states for every component in the system, determining the control variables or attribute values in each submodule, and depicting the statechart of each submodule.
3. Compose the statecharts of the components into a larger statechart using serial, union, parallel, or merging compositions based on events or states shared.
4. Repeat the above process until there are no isolated statecharts.
5. Mark the initial configuration of the system.
6. Convert the entire statechart into PLC programs based on the transformation rules.

PLC programs are produced through the overall statechart model. The transformation from statecharts to PLC notations will be described in the next subsection.

2.4.2 Transformation Rules of Statecharts and PLC in HSM

System development can be taken as a series of transformations that transform system specifications from one language to another [28]. The transformation rules of HSM map every state, event, and action in the statechart into the corresponding PLC domain. The output of the transformation is a ladder diagram (LD) or a set of control logic equations. An LD is a conventional representation for a PLC application. An LD containing rungs of symbolic entries expresses the control logic. Most symbolic entries correspond to I/O contacts, internal relays, counters, and timers.

However, there are various syntaxes or types in the existing PLC languages. The general form of instruction with “set” and “reset” was used in this study. The instruction “set” will maintain the element “on”, irrespective of the control circuit being “on” or “off”. The instruction “reset” will turn the element “off”. The transformation rules are as follows:

Rule 1: basic mapping. The next state is set on, if the previous state has been satisfied and the input event has occurred. In Fig. 5, $B(1) = Ax$ indicates that state A and event x are the conditions to set state B on.

Rule 2: serial composition mapping. In Fig. 5, $B(1) = A$ indicates that state A is the condition to set state B on, while $A(0) = B$ indicates that state B is the condition to reset state A.

Rule 3: parallel composition mapping. In Fig. 5, t_1 and t_3 are parallel states. Because trigger events t_1 and t_3 are parallel transitions, x is the common event. Thus, the corresponding Boolean Eq. are as follows: $B(1) = xAD$, $E(1) = xAD$, $A(0) = B$, $D(0) = E$.

Rule 4: merging composition mapping. In Fig. 5, states B and E have identical input and output events $\{x, u\}$. A new state B' can be defined after merging the states $\{B, E\}$.

Rule 5: union composition mapping. In Fig. 6, $S(1) = Rt_1$, $T(1) = Rt_2$, and $U(1) = Rt_3$ indicates that state R and event t_1 (respectively, t_2 and t_3) are the conditions to set state S (respectively, T and U) on.

3. Design Example of HSM

In this section, we will employ a machining workcell as an example to illustrate the systematic steps of the HSM.

3.1 Introduction to a Workcell Example

The workcell example consists of a machining module and a material-handling module. The machining module contains two identical machines M_1 and M_2. The material-handling module comprises a robot, an input buffer, and an output buffer. The workcell controller coordinates the actions of M_1, M_2, the robot, and buffers. The equipment layout for the workcell example is shown in Fig. 7.

When a part moves into the input buffer, the workcell process in this example is divided into six steps:

1. Start loading the part in M_1 on M_2 (if a machine is ready) from the input buffer using the robot.
2. Stop loading task.
3. Start machining on the corresponding machine.
4. Stop machining task.
5. Start unloading the part into the output buffer using the robot.
6. Stop unloading task and reset the corresponding machine, output buffer, and robot.

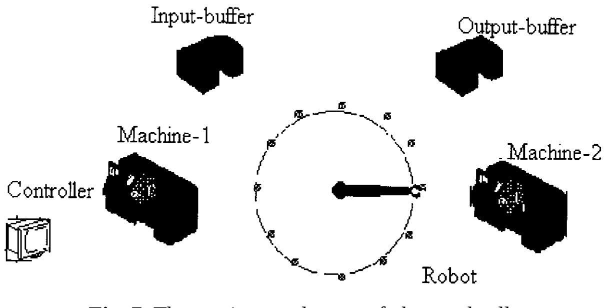


Fig. 7. The equipment layout of the workcell.

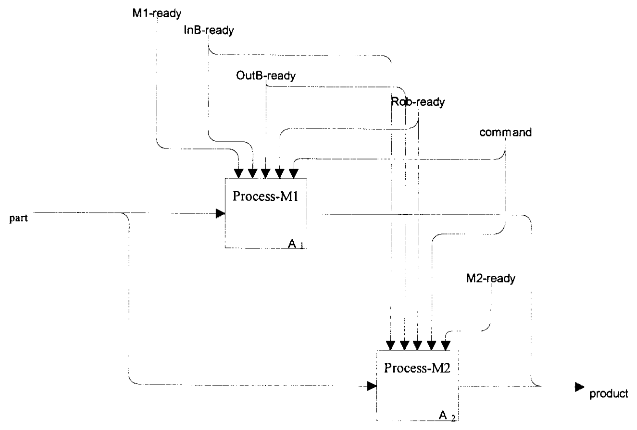


Fig. 8. Top-level functional specifications of the workcell example.

At each step of the processing cycle, it is important to identify the trigger events since these events determine the state changes and module compositions.

3.2 Statecharts of the Workcell Example

According to the design steps and transformation rules mentioned in Section 2.4, the procedures to construct the workcell statecharts for this example are illustrated in the following subsections.

3.2.1 Depict the Functional Hierarchy of the Overall System

The top-level requirement specification is given in Fig. 8 using IDEF0 notations. The conditions for changing states can be identified by their input, control, and output relationships. This information will be applied to construct the statechart for each element of the workcell.

Detailed functional specifications for machine M_1 are refined in Fig. 9. According to these functions and flow information, trigger conditions can be identified in the statecharts.

Detailed functional specifications for machine M_2 are similar to M_1 and are depicted in Fig. 10.

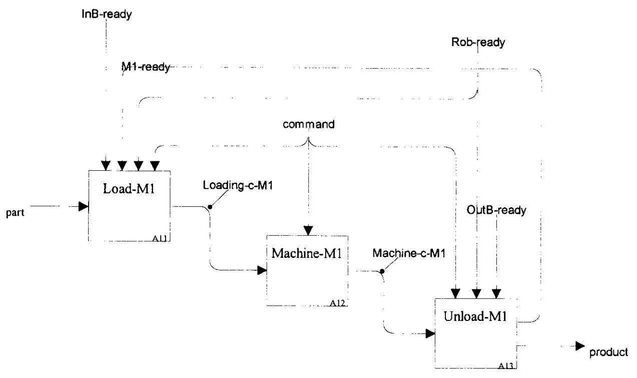


Fig. 9. Functional specifications for machine M_1.

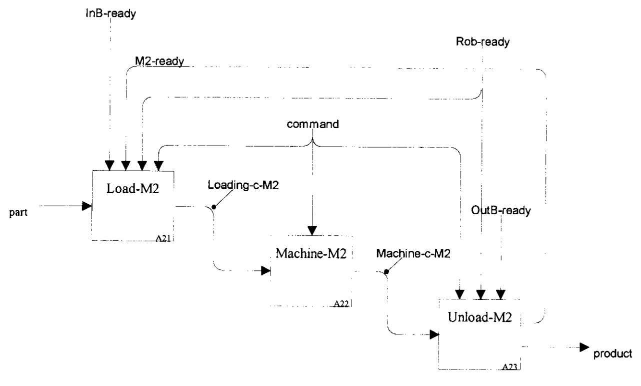


Fig. 10. Functional specifications for machine M_2.

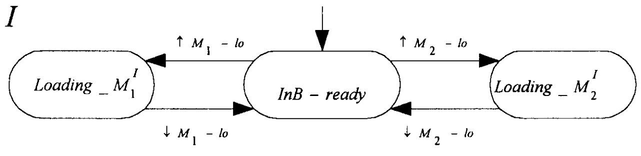


Fig. 11. The statechart of the input buffer.

3.2.2 Construct the Statechart of Each Submodule

To construct the corresponding statechart of a submodule, the acceptable states are analysed first. Trigger events in a basic statechart connect these states. The details of each submodule are then drawn step-by-step. For instance, the status of the input buffer includes ready and busy states. The statechart of the input buffer can be expressed as in Fig. 11. The busy state of the input buffer indicates that the parts are in the process of loading on machines M_1 or M_2. The leading edge in ↑(M_1-*lo*) denotes that M_1 starts loading parts. The falling edge in ↓(M_1-*lo*) denotes that M_1 finishes loading parts and returns the input buffer back to ready state. The state "Loading-*c*-M_1" denotes that the loading task of M_1 is completed.

The statechart of machine M_1 is presented in Fig. 12, where ↑(M_1-*ma*) denotes that M_1 starts machining a part and ↓(M_1-*ma*) denotes that M_1 finishes machining a part.

The statechart of machine M_2 is similar to that of M_1 and is displayed in Fig. 13.

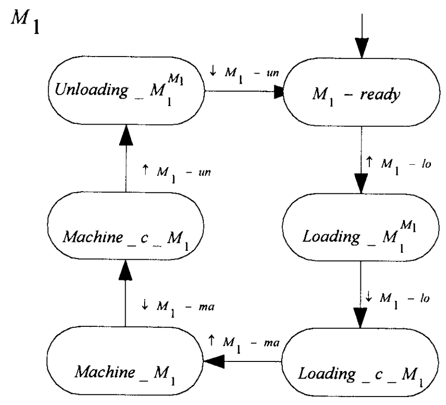


Fig. 12. The statechart of machine M_1.

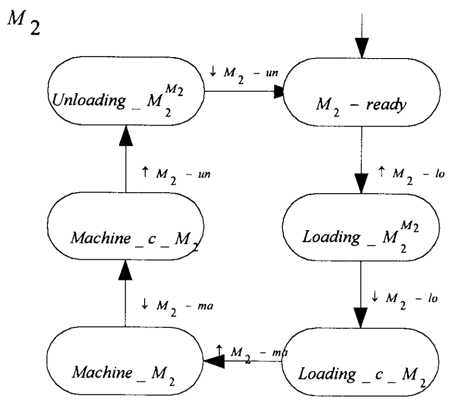


Fig. 13. The statechart of machine M_2.

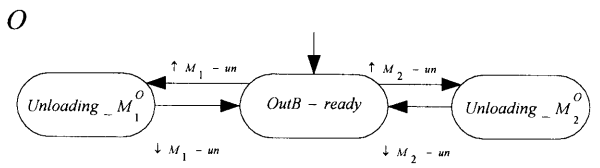


Fig. 14. The statechart of the output buffer.

The statechart of the output buffer is shown in Fig. 14, where ↓(M_1-*un*) denotes that M_1 finishes unloading a part, and the part is moved out. Consequently, the output buffer accepts the instruction to become ready.

The processing status of the robot contains five states: Rob-ready, Loading-M_1, Loading-M_2, Unloading-M_1, and Unloading-M_2. The robot statechart is shown in Fig. 15. According to union composition, the Rob-ready state is shared by four transitions: ↑M_1-*lo*, ↑M_2-*lo*, ↑M_1-*un* and ↑M_2-*un*.

3.2.3 Compositional Statecharts

We can combine two statecharts of the workcell example using the principle of parallel composition. For simplicity, we substitute the complicated symbols with simple characters, as in Table 1, where * denotes {*I*, *O*, M_1, M_2, *R*}. There are three types of event in the system, i.e. external events (e.g. initialise), start events (e.g. ↑(M_1-*lo*)), and stop events (e.g. ↓(M_1-*lo*)). Events determine the compositional semantics.

For instance, as a trigger event ↑(M_1-*lo*) occurs, this event will also trigger the action of loading in the robot and the input buffer.

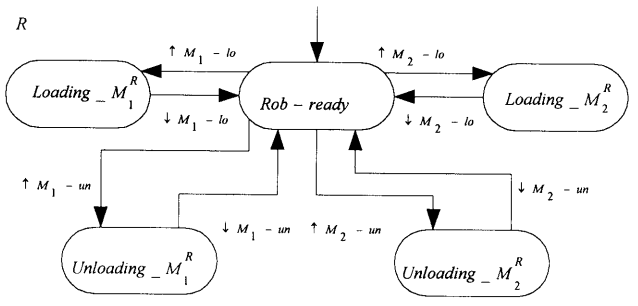


Fig. 15. The statechart of the robot.

Table 1. Event and state abbreviation for the workcell example.

Events		States	
e_1	↑(M_1-*lo*)	S_1	M_1-ready
e_2	↓(M_1-*lo*)	S_2*	Loading-M_1*
e_3	↑(M_1-*ma*)	S_3	Loading-*c*-M_1
e_4	↓(M_1-*ma*)	S_4	Machining-M_1
e_5	↑(M_1-*un*)	S_5	Machining-*c*-M_1
e_6	↓(M_1-*un*)	S_6*	Unloading-M_1*
e_7	↑(M_2-*lo*)	S_7	M_2-ready
e_8	↓(M_2-*lo*)	S_8*	Loading-M_2*
e_9	↑(M_2-*ma*)	S_9	Loading-*c*-M_2
e_{10}	↓(M_2-*ma*)	S_{10}	Machining-M_2
e_{11}	↑(M_2-*un*)	S_{11}	Machining-*c*-M_2
e_{12}	↓(M_2-*un*)	S_{12}*	Unloading-M_2*
		S_{13}	InB-ready
		S_{14}	OutB-ready
		S_{15}	Rob-ready

3.2.4 Continue Composing Until there is no Isolated Statechart

According to the parallel composition of statecharts, the final statechart of this workcell controller can be obtained as shown in Fig. 16. The system is always in these five states {*I*, *O*, M_1, M_2, *R*}. Furthermore, there are four merging states: $s_2 = [s_2^I, s_2^{M_1}, s_2^R]$, $s_6 = [s_6^O, s_6^{M_1}, s_6^R]$, $s_8 = [s_8^I, s_8^{M_2}, s_8^R]$, and $s_{12} = [s_{12}^O, s_{12}^{M_2}, s_{12}^R]$ in terms of merging composition.

3.2.5 Mark the Initial Configuration

When the resources are available, the final statechart is marked with the initial configuration. In Fig. 16, s_0 denotes that system power is on. We assume that the initial states are ready after initialising the equipment. The initial configuration *C* equals $\{s_1, s_7, s_{13}, s_{14}, s_{15}\}$.

3.3 PLC Control Equation

After specifications having been developed and analysed, the control logic of a controller can be obtained based on the transformation rules. The PLC control logic Eq. corresponding to statecharts are given below. The corresponding ladder diagram is also shown in Fig. 17.

$$s_1(1) = e_6\ s_6; \quad s_1(0) = s_2$$
$$s_2(1) = e_1\ s_1\ s_{13}\ s_{15}; \quad s_2(0) = s_3\ s_{13}\ s_{15}$$
$$s_3(1) = e_2\ s_2; \quad s_3(0) = s_4$$
$$s_4(1) = e_3\ s_3; \quad s_4(0) = s_5$$
$$s_5(1) = e_4\ s_4; \quad s_5(0) = s_6$$
$$s_6(1) = e_5\ s_5\ s_{14}\ s_{15}; \quad s_6(0) = s_1\ s_{14}\ s_{15}$$
$$s_7(1) = e_{12}\ s_{12}; \quad s_7(0) = s_8$$
$$s_8(1) = e_7\ s_7\ s_{13}\ s_{15}; \quad s_8(0) = s_9\ s_{13}\ s_{15}$$
$$s_9(1) = e_8\ s_8; \quad s_9(0) = s_{10}$$
$$s_{10}(1) = e_9\ s_9; \quad s_{10}(0) = s_{11}$$
$$s_{11}(1) = e_{10}\ s_{10}; \quad s_{11}(0) = s_{12}$$
$$s_{12}(1) = e_{11}\ s_{11}\ s_{14}\ s_{15}; \quad s_{12}(0) = s_7\ s_{14}\ s_{15}$$
$$s_{13}(1) = e_2\ s_2 + e_8\ s_8; \quad s_{13}(0) = s_8 + s_2$$

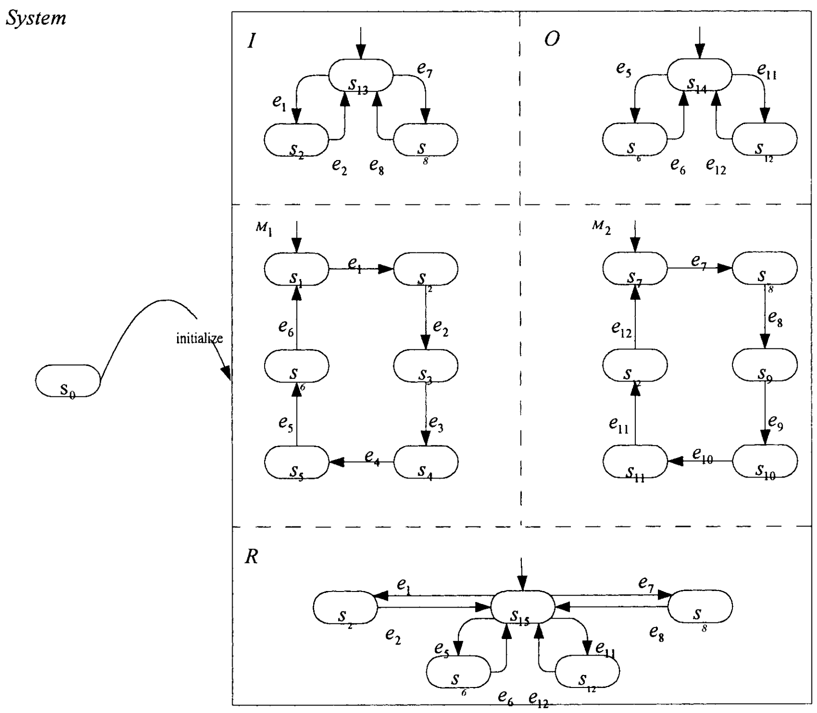


Fig. 16. The statechart of the workcell example.

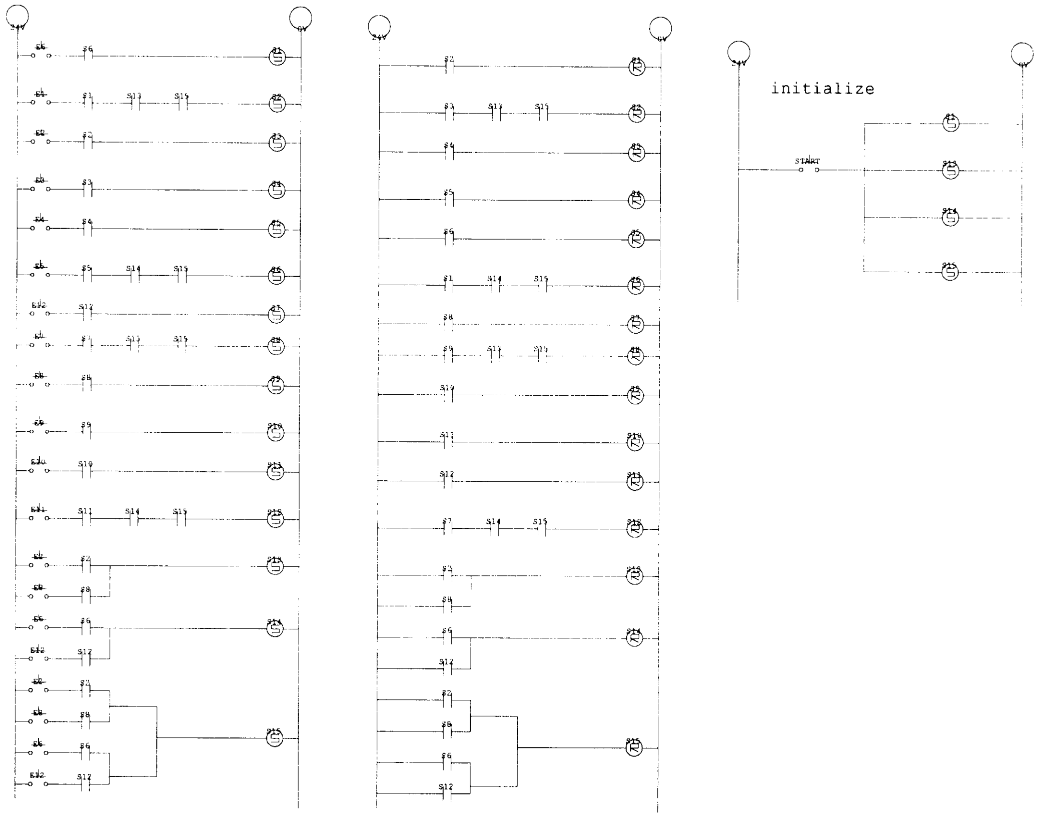


Fig. 17. Ladder diagram.

$s_{14}(1) = e_6\ s_6 + e_{12}\ s_{12};\ s_{14}(0) = s_6 + s_{12}$
$s_{15}(1) = e_2\ s_2 + e_6\ s_6 + e_8\ s_8 + e_{12}\ s_{12};\quad s_{15}(0) = s_2 + s_6 + s_8 + s_{12}$

4. Static Structure and Behaviour of Statecharts

After the introduction to the statecharts, the formal syntax and some related properties will be described in this section.

4.1 Static Structure of Statecharts

The following definitions are taken and modified from Pnueli and Shalev [29]. These definitions are used to describe the behaviour of a workcell. A statechart M can be described by a 6-tuple (S, $\leq$, $\sim$, V, c_0, F) model. Each tuple is described as follows:

S is a finite set of states. A certain subset of S is defined as a configuration, which denotes the consistent combination of states, i.e. the maximal set of concurrent states. The set of all configurations composes a "Config".

$\leq$ is a parent/child relation, where $x \leq y$ refers to the fact that y is a parent of x or x is y, while $x < y$ refers to the fact that y is a parent of x strictly. $\sigma(y)$ refers to the set of all descendant states of y.

$\sim$ defines the relation of parallelism on states, where $x \sim y \Rightarrow \exists z\ x, y \in \sigma\ (z)$.

V is the set of I/O history in statecharts. To simplify the presentation, let V equal Φ primarily, where Φ denotes an empty set.

c_0 is the initial global state, where a global state $(c, v) \in$ *Config* $\times V$

For instance, the static model of statecharts of the workcell in Fig. 16 can be denoted as (S, $\leq$, $\sim$, V, c_0), where $S = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}, s_{11}, s_{12}, s_{13}, s_{14}, s_{15}\}$, $I = \{s_2, s_8, s_{13}\}$, $M_1 = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7\}$, $M_2 = \{s_7, s_8, s_9, s_{10}, s_{11}, s_{12}\}$, $O = \{s_6, s_{12}, s_{14}\}$, and $R = \{s_2, s_6, s_8, s_{12}, s_{15}\}$.

Since $s_1 \leq M_1$ and $s_2 \leq M_1$, the relation $s_1 \sim s_2$ can be obtained.

Some global states can be expressed as follows: $c_0 = (\{s_0\}, \Phi)$ and $c_1 = \{s_1, s_7, s_{13}, s_{14}, s_{15}\}$, *initialise*).

4.2 Behavioural Analysis

The mapping function F is a mapping between global states F: $C \rightarrow C$, which describes the behaviour of statecharts. The execution steps of statecharts represent the dynamic behaviour of a system. For instance, the execution steps for machine M_1 are described as follows.

1. A new part is put into the input buffer. The state values of M_1, the buffer, and the robot is "ready". The configuration of the global state is $\{s_1, s_7, s_{13}, s_{14}, s_{15}\}$.
2. As event e_1 occurs, the robot loads the part on M_1. The global state is $\{s_2, s_7, s_{14}\}$.
3. As event e_2 occurs, the robot finishes loading the parts. The configuration of the global state is $\{s_3, s_7, s_{13}, s_{14}, s_{15}\}$.
4. As event e_3 occurs, the machine M_1 starts machining. The configuration of the global state is $\{s_4, s_7, s_{13}, s_{14}, s_{15}\}$.
5. As event e_4 occurs, the machine M_1 finishes machining. The configuration of the global state is $\{s_5, s_7, s_{13}, s_{14}, s_{15}\}$.
6. As event e_5 occurs, the robot takes the part out of the machine M_1. The parts are unloaded. The configuration of the global state is $\{s_6, s_7, s_{13}\}$.
7. As event e_6 occurs, the robot finishes the unloading task. The system is reset. The configuration of the global state is $\{s_1, s_7, s_{13}, s_{14}, s_{15}\}$.

As mentioned in the preceding section, the sequence of global states or the process from the initial configuration to the final configuration denotes the system dynamic behaviour. However, there are still some unsolved problems, such as, how to avoid abnormalities in the dynamic behaviour of HSM. We adopted the following definitions and theorem proposed by Jahanian and Mok [23] to guarantee the absence of abnormalities.

Definition 4.1. Transition cycle. For any transition t_i and state S_i, $1 \leq i \leq n$, when t_i leaves S_i, the system becomes S_{i+1}. Also, when t_n leaves S_n, the system becomes S_1. This indicates that T (t_1, …, t_i, …, t_n) is a transition cycle of state set {S_1, …, S_i, …, S_n).

Definition 4.2. Abnormality of transition is defined as follows. If a transition between two states is instantaneous, then a transition cycle occurring at the same time will result in an abnormality.

Theorem 4.3. The transition cycle proposed by the statechart model of the workcell in Section 3 does not result in an abnormality.

Proof. The workcell example in Section 3 illustrates that the working states of the physical equipment consist of loading, machining, and unloading. It takes a period of time to finish the task in each state. It is impossible for a transition cycle to immediately occur. Thus, this system is free from abnormality.

5. Discussion and Related Work

In this section, we explore some statechart properties and discuss the closely related work, e.g. Petri nets. A Petri net is the most frequently used tool for manufacturing systems modelling. Statecharts and Petri nets have equivalent representative capabilities because they are both state-based models [14]. For instance, a transition in a statechart is labelled with an input event and an output action. This relationship can be translated into Petri net notation, where "e_1 = on" is viewed as an input state in a Petri net, and "a_1 = on" is viewed as an output state. It is assumed that the reader is familiar with Petri net notation [30,31]. Figure 18 relates these two state-based models. After having translated a statechart into a Petri net, we can investigate system properties using the Petri net, e.g. invariant analysis.

In the Petri net model, invariant analysis is an often-used method, which allows a designer to explore the logical properties of a target system in a formal way. A place invariant expresses the conservative property using a weighted set of tokens. A place invariant is defined as: $i^T\mu = i^T\mu_0$, where i^T is an invariant vector, μ is a marking of the system, and μ_0 is an initial marking of the system. Invariant vectors can be obtained by the equation, $i^TA = 0$, where A is an incidence matrix.

The workcell example in Section 3 can be translated into Petri net notation, as shown in Fig. 19. Its incidence matrix is as follows.

$$A = \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \\ -1 & 1 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & -1 & 1 \\ -1 & 1 & 0 & 0 & -1 & 1 & -1 & 1 & 0 & 0 & -1 & 1 \end{bmatrix}$$

According to the constraint equation: $i^TA = 0$, the following invariant vectors can be obtained.

$$i_1^T = (1\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0) \quad (1)$$
$$i_2^T = (0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0) \quad (2)$$
$$i_3^T = (0\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0) \quad (3)$$
$$i_4^T = (0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0) \quad (4)$$
$$i_5^T = (0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 1) \quad (5)$$

According to the invariant vectors, the following marking relations can be yielded.

$$\mu(s_1) + \mu(s_2) + \mu(s_3) + \mu(s_4) + \mu(s_5) + \mu(s_6) = 1 \quad (6)$$
$$\mu(s_7) + \mu(s_8) + \mu(s_9) + \mu(s_{10}) + \mu(s_{11}) + \mu(s_{12}) = 1 \quad (7)$$
$$\mu(s_2) + \mu(s_8) + \mu(s_{13}) = 1 \quad (8)$$
$$\mu(s_6) + \mu(s_{12}) + \mu(s_{14}) = 1 \quad (9)$$
$$\mu(s_2) + \mu(s_6) + \mu(s_8) + \mu(s_{12}) + \mu(s_{15}) = 1 \quad (10)$$

The invariant vector properties indicate that there are not two states, s_i and s_j, $s_i \neq s_j$, possessing the same token simultaneously. In a manufacturing system, this property reflects the fact that resources are always constrained. These conservative properties have been demonstrated as the OR-state from Figs 11 to 15. Additionally, the following properties can be obtained.

1. The machine M_1 stays at just one state in any period of time (OR-state of statecharts). Hence, the total marking equals one, which is displayed in Eq. (6).

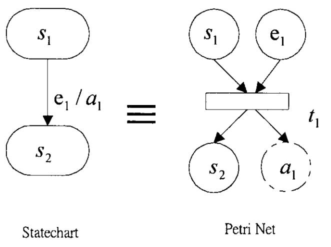


Fig. 18. The corresponding notations between statecharts and Petri nets.

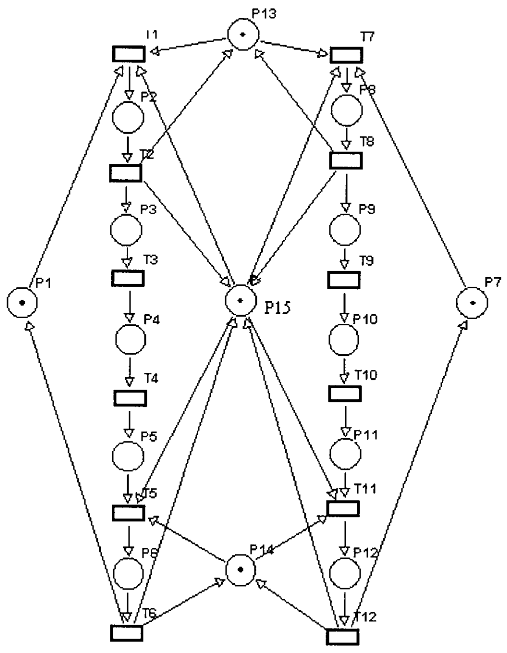


Fig. 19. The Petri net of the workcell example in Section 3.

2. Similarly, machine M_2 stays at just one state in any period of time. Therefore, the total marking equals one, which is expressed in Eq. (7).
3. The input buffer stays at just one state {InB-ready, Loading-M_1, Loading-M_2} in any period of time. Hence, the total marking equals one, which is shown in Eq. (8).
4. The output buffer stays at just one state of {OutB-ready, Unloading-M_1, Unloading-M_2} in any period of time. Thus, the total marking equals one, which is displayed in Eq. (9)
5. The robot stays at just one state {Rob-ready, Loading-M_1, Loading-M_2, Unloading-M_1, Unloading-M_2} in any period of time. Hence, the total marking equals one, which is indicated in Eq. (10).

Since the OR-state in statecharts represents the conservative property of a submodule, a consistent result can be obtained using Petri net invariant analysis. This property demonstrates the viability of the HSM for developing a workcell controller.

Unlike the Petri net, another related method of workcell controller design using the objected-oriented technique is a semi-formal approach. For instance, Lai and Lee [32] employed objected-oriented static and dynamic models to specify and design a controller. Liu et al. [33] presented an integrated reference architecture using object-oriented analysis and design methods to develop a shop-floor control system. These approaches are easy for modelling and designing a target system, but lack a formal foundation.

6. Conclusions

In this paper, we present an HSM for the design of a workcell controller in a manufacturing system. The HSM is a hybrid specification method, which captures the advantages of both top-down and bottom-up approaches and helps to transform statecharts into PLC programs directly. The graphical notations of IDEF0 and statecharts enhance the requirement apprehension and facilitate the transformation of the PLC programs. One of the main ideas is to keep the requirement specifications simple and easy to implement. The integration of functional and behavioural specifications provides engineers with a good starting point in designing a workcell controller.

The HSM is illustrated using an example of a workcell controller. The result is consistent with Petri net analysis. The relationships between the place invariant and conservative properties are also demonstrated in this study.

The practical importance of the HSM can be concluded as follows:

1. It specifies the functional and behavioural requirements of a system in a diagrammatic form, which has the virtue of visual description that makes the specification easy to understand.
2. It develops a workcell controller through integrating the top-down decomposition and the bottom-up composition approaches.
3. It provides systematic steps to analyse and design the workcell controller.
4. It provides some transformation rules between statechart notations and PLC programs.
5. It is consistent with the Petri net approach.

In the future, we plan to enrich the HSM with the capability to verify in temporal logic, and to apply the HSM to more complicated systems.

References

1. S. Caselli, C. Papaconstantinou, K. Doty and S. Navathe, "A structure-function-control paradigm for knowledge-based modeling and design of manufacturing workcells", Journal of Intelligent Manufacturing, 3, pp. 11–30, 1992.
2. A. Bauer, R. Bowden, J. Browne, J. Duggan and G. Lyons, Shop Floor Control Systems from Design to Implementation, Chapman and Hall, London, 1994.
3. S. Weygandt, "Getting the MES model – methods for system analysis", ISA Transactions, 35(2), pp. 95–103, 1996.
4. M. S. Mujtaba, "Simulation modeling of a manufacturing enterprise with complex material, information and control flows", International Journal of Computer Integrated Manufacturing, 7(1), pp. 29–46, 1994.
5. W. H. Jeng and G. R. Liang, "An automated manufacturing system design methodology based on a triple flow model", International Journal of Industrial Engineering, 5(1), pp. 17–27, 1998.
6. M. A. Jafari and T. O. Boucher, "A rule based system for generating a ladder logic control program from a high-level systems model", Journal of Intelligent Manufacturing, 5, pp. 103–120, 1994.
7. G. R. Liang and H. M. Hong, "Hierarchy transformation method to manufacturing system specification, design, verification, and implementation", Computer-Integrated Manufacturing System, 7, pp. 191–205, 1994.
8. S. G. Shanmugham and C. A. Roberts, "Application of graphical specification methodologies to manufacturing control logic development: a classification and comparison", International Journal of Computer Integrated Manufacturing, 11(2), pp. 142–152, 1998.
9. W. Wang, K. Popplewell and R. Bell, "An integrated multi-view system description approach to approximate factory modelling", International Journal of Computer Integrated Manufacturing, 6(3), pp. 43–53, 1993.
10. FIPS 183, Integration Definition for Function Modeling (IDEF0), National Institute of Standards and Technology, 1993.
11. M. Uzam and A. H. Jones, "Discrete event control system design using automation Petri nets and their ladder diagram implementation", International Journal of Advanced Manufacturing Technology, 14, pp. 716–728, 1998.
12. V. Botta, A. Guinet and D. Boulle, "Object-oriented analysis with structured and integrated specifications and solutions (OASISS) for production system control", Journal of Intelligent Manufacturing, 8, pp. 3–14, 1997.
13. A. K. Chakravarty, H. K. Jain, J. J. Liu and D. L. Nazareth, "Object-oriented domain analysis for flexible manufacturing systems", Integrated Computer Aided Engineering, 4, pp. 290–309, 1997.
14. G. Bucci, M. Campanai and P. Nesi, "Tools for specifying real-time systems", Real-Time Systems, 8, pp. 117–172, 1995.
15. M. P. E. Heimdahl and N. G. Leveson, "Completeness and consistency in hierarchical state-based requirements", IEEE Transactions on Software Engineering, 22(6), pp. 363–377, 1996.
16. J. C. Marty, A. E. K. Sahraoui and M. Sartor, "Statecharts to specify the control of automated manufacturing systems", International Journal of Production Research, 36(11), pp. 3183–3215, 1998.
17. H. P. Barendregt, The Lambda Calculus – Its Syntax and Semantics, North-Holland, 1984.
18. G. R. Liang, "A lambda calculus model for the officer-cell phenomenon in manufacturing systems", Proceedings of the 6th International Conference of CAD/CAM, Robotics and Factories of the Future, London, UK, August 1991.
19. D. Harel, "Statecharts: a visual formalism for complex systems", Science of Computer Program, 8, pp. 231–274, 1987.
20. D. Harel, H. Lachover and A. Naamad, "Statemate: a working environment for the development of complex reactive systems", IEEE Transactions on Software Engineering, 16(4), pp. 403–414, 1990.
21. D. Coleman, F. Hayes and S. Bear, "Introducing objectcharts or how to use statecharts in object-oriented design", IEEE Transactions on Software Engineering, 18(1), pp. 9–18, January 1992.
22. Y. Brave and M. Heymann, "Control of discrete event systems modeled as hierarchical state machines", IEEE Transactions on Automatic Control, 38(12), pp. 1803–1819, 1993.
23. F. Jahanian and A. K. Mok, "Modechart: a specification language for real-time systems", Transactions on Software Engineering, 20(12), pp. 933–947, 1994.
24. A. Sowmya and S. Ramesh, "Extending statecharts with temporal logic", IEEE Transactions on Software Engineering, 24(3), pp. 216–231, December 1998.
25. A. W. Naylor and R. A. Volz, "Design of integrated manufacturing system control software", IEEE Transaction on Systems, Man and Cybernetics, 17(6), pp. 881–897, 1987.
26. R. David, "Grafcet: a powerful tool for specification of logic controllers", IEEE Transactions on Control Systems Technology, 3(3), pp. 253–268, September 1995.

27. J. R. Polland, “Ladder logic remains the PLC language of choice”, Control Engineering, pp. 77–79, April 1994.
28. H. Mili, F. Mili and A. Mili, “Reusing software: issues and research directions”, IEEE Transactions on Software Engineering, 21(6), pp. 528–562, June 1995.
29. A. Pnueli and M. Shalev, “What is in a step: on the semantics of statecharts”, Lecture Notes in Computer Science, 526, pp. 244–264, 1991.
30. J. L. Peterson, Petri-Net Theory and Modeling of Systems, Prentice-Hall, Englewood Cliffs, NJ, 1981.
31. T. Murata, “Petri net: properties, analysis, and applications”, Proceedings IEEE, 77, pp. 541–580, 1989.
32. H. F. Lai and C. E. Lee, “Integrating the object-oriented static and dynamic models to design a controller of cellular manufacturing system”, Journal of Science and Technology, 8(3), pp. 215–224, July 1999.
33. C. M. Liu, C. F. Chien and L. Y. Ho, “An object-oriented analysis and design method for shop floor control system”, International Journal of Computer Integrated Manufacturing, 11(5), pp. 379–400, 1998.