An Extended Cellular Automaton Model for Flexible Manufacturing Systems

H.C. Shen¹ H.L. Chau K.K. Wong

Department of Computer Science

Hong Kong University of Science and Technology

Clear Water Bay

Hong Kong

Abstract

In this paper, we propose a high-level state-based dependency description and structuring formalism for flexible manufacturing systems (FMS). This **extended cellular automaton model (ECAM)** is designed to capture the behaviour and performance of a large complex concurrent system in an easier and more concise way than is possible with other contemporary FMS models. Concurrency problems of FMS can be clearly identified by the model.

The model is defined mathematically as a quintuple by adapting the cellular automaton. A set of graphical symbols is also defined to represent the states of the model. The proposed model can serve as a formal specification and documentation tool for an FMS. It can also provide a basis for modelling important characteristics

---

¹On leave from the Dept.of Systems Design Engineering, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1.

of FMS which includes conflict resolution, dependencies and starvation of resources. In real applications, it can serve as a conceptual model in the FMS design process. Based on this model, a prototype system has been developed to generate a set of executable grammar rules. With appropriate extensions, the system can be used as simulation and performance analysis tools.

Keywords: Flexible Manufacturing Systems; Cellular Automaton; Modelling

1 Introduction

Flexible manufacturing systems (FMS) have become larger in size and more complicated in its operations. In order to investigate the underlying principles of such systems, an appropriate modelling tool is necessary. Modelling the complex interactions in an FMS is a formidable task. Markovian models, queueing network models and simulation techniques have been quite useful in the analysis, planning, and control of FMS operations. The perturbation analysis technique (a hybrid analytical-simulation methodology) has been used for the optimization of FMSs. The modelling power of Petri nets is well-known by considering models of operating systems, chemical reactions, computer programs, computer systems, databases, communication protocols, and recently, models of FMS. The research papers of all these modeling tools have been collected by Desrochers[5]. One of the disadvantages of these models, except the Petri net model, is that they all require much knowledge in mathematics and statistics. In the Petri net model, users have to identify all the states of FMS, transition linkages and the correct numbers of tokens and their corresponding positions, which are far more complicated than users expected if they only want to construct a conceptual model for an FMS.

In [7], Shen and Yan have proposed a finite automaton model to represent a single agent in an autonomous assembly workcell. It preserves the conciseness and expressiveness of

finite automaton theory. However, finite automaton does not model the inter-relationships and dynamics between agents in an FMS. Therefore, in [8], they proposed a model based on the cellular automaton theory which is an extension of finite automaton model to represent an FMS.

Cellular automata were first proposed by Von Neumann in the late forties [9]. Cellular automata have a wide variety of applications in biological systems [1], physical chaotic systems, pattern recognition and computational theory [6]. The original cellular automaton theory was developed for homogeneous systems. Shen and Yan [8] extended it to include heterogeneous features which are typical in FMS. The extended model is ideal to capture some important characteristics of FMS. However, there are some situations which cannot be represented by their model. In this paper, we shall address some of these problems by using the proposed model.

It has also been demonstrated in [3] that this model can be mapped to constraint narrowing grammar rules. Thus, a prototype system has been designed and developed to generate these executable rules. With appropriate extensions, it can be used as simulation and performance analysis tools.

2 Extended Cellular Automaton Model (ECAM)

We shall adapt the definition of the cellular automaton model proposed in [8]. The transition function is extended to include influences to neighboring cells, i.e. transition can be caused by states from neighboring cells and/or can cause state changes in the neighboring cells simultaneously. The previous model allows transition to cause state change in one cell only.

2.1 Mathematical Definition

Definition 1 A Cellular Space, C , consists of a finite set of cells, $\{c_1, c_2, \dots, c_N\}$, where N is the number of cells in the space.

Definition 2 A cell c_i is a deterministic finite automaton defined by a quintuple $(Q_i, \sigma_i, \delta_i, q_o^{(i)}, F_i)$, where

- Q_i is a finite set of states, $\{q_o^{(i)}, q_1^{(i)}, \dots, q_{n_i}^{(i)}\}$, where n_i+1 is the number of states of cell c_i . Cells represent resources and states represent the status the resource associated with this cell.
- σ_i is a finite alphabet of state symbols, $\sigma_i = \{a_1^{(i)}, a_2^{(i)}, \dots, a_{s_i}^{(i)}\}$, where s_i is the number of symbols of cell c_i . These symbols are used to control the transition from state to state.
- δ_i is a transition function which determines the state of the automaton at the next time step. Each cell has a neighborhood which consists of itself and a number of neighbors. A neighbor of a cell c_i is another cell in the cellular space C and has some relationship with c_i . The state of the cell, c_i , at the next instant $(t+1)$ is uniquely determined by the states of its neighbors and their corresponding state symbols at instant t . Let us denote the set of neighborhood of cell c_i by the set $G_i = \{c_1^{(i)}, c_2^{(i)}, \dots, c_{m_i}^{(i)}\}$, and the corresponding sets of state symbols of its neighbors by the set $D_i = \{\sigma_1^{(i)}, \sigma_2^{(i)}, \dots, \sigma_{m_i}^{(i)}\}$, where $m_i - 1$ is the number of neighbors. Let $H_i = Q_1^{(i)} \times Q_2^{(i)} \times \dots \times Q_{m_i}^{(i)}$, and $S_i = \sigma_1^{(i)} \times \sigma_2^{(i)} \times \dots \times \sigma_{m_i}^{(i)}$, where $Q_j^{(i)}$ and $\sigma_j^{(i)}$ are the corresponding sets of states and state symbols of $c_j^{(i)}$ respectively, and $j = 1, 2, \dots, m_i$, then δ_i can be defined as

$$\delta_i : H_i \times S_i \longrightarrow H_i$$

- $q_o^{(i)}$ is the initial state of the cell c_i . It is the initial set up of the resources.

- F_i is the set of final states that the automaton may stop when it reaches a state in F_i .

Definition 3 An Extended Cellular Automaton is defined as a quintuple, $(P, \Sigma, \phi, p_o, \mathcal{F})$, where

- P is the set of configurations. A configuration is an allowable assignment of states to all cells in the cellular space. P can be represented as $P = Q_1 \times Q_2 \times \dots \times Q_N$, where Q_i is the set of states of cell c_i for $i = 1, 2, \dots, N$ and N is the number of cells in the cellular space.
- Σ is the set of input words of the cellular automaton. An input word of the cellular automaton is a suitable assignment of input symbols of all cells. Σ can be represented as $\Sigma = \sigma_1 \times \sigma_2 \times \dots \times \sigma_N$, where σ_i is the set of input symbols of cell c_i for $i = 1, 2, \dots, N$ and N is the number of cells in the cellular space.
- ϕ is the global transition function. It is a transition function from one configuration to another, $\phi : P \times \Sigma \longrightarrow P$.
- p_o is the initial configuration of the cellular automaton.
- $\mathcal{F}$ is the set of final configuration.

In the model, each cell is considered as an atomic unit which can only have one current state at any time instant. Therefore, each cell contains one-and-only-one current state symbol at any time instant.

2.2 Definition of Graphical Symbols

To facilitate an FMS engineer's ease in using the model in design and to represent the states of the FMS at different time instant, a set of graphical symbols is defined to represent the model (Figure 1). This is NOT a one-to-one mapping from the mathematical definition.

Figure 1: Graphical Representation

1. a *cell* is represented by a dash-line rectangle;
2. a double directional arrow represents neighbor relationship between two cells;
3. a thick vertical bar together with single directional arrows represents a *transition*;
4. a *state* is represented by a solid-line rectangle;
5. an * symbol at the upper-left corner inside a solid rectangle represents the initial state of a cell;
6. an ^ symbol at the upper-right corner inside a solid rectangle represents a final state of a cell;
7. an @ symbol inside the solid rectangle represents the current state of a cell;
8. numbers are used to indicate the priority of transition if the transition affects more than one cell. Here we arbitrarily assume the lower number has higher priority. Priority may relate to the current state, i.e. depending on the current state, priority of the transition may be different.

Figure 2: The ECAM of An Assembly System

3 Capabilities of the Model

We shall demonstrate some capabilities of the proposed extended cellular automaton model. A simple example of an assembly system will be used as the platform. This assembly system consists of (a) two robots which carry out the assembly tasks; (b) one bin which contains parts to be assembled and (c) an error identification and recovery system which performs error identification and recovery whenever the robots encounter problems. Figure 2 is the graphical representation of this system with four cells: **Robot1**, **Robot2**, **Bin** and **Iden_Reco**. **Bin** and **Iden_Reco** are shared between **Robot1** and **Robot2**. Both will reach **Bin** for parts to be assembled. To avoid collision of the robots, **Bin** can be “occupied” by only one robot at any time instant.

When transitions involve more than one cell, we shall differentiate local and non-local, direct and indirect transitions.

Definition 4 *A transition is local with respect to cell, c_i if it causes change of state to cell c_i only. Otherwise, the transition is non-local.*

Definition 5 *A direct local transition is a local transition which does not depend on*

Figure 3: Local vs Non-local and Direct vs Indirect Transitions

the states of any of its neighbors. Otherwise, if a local transition depends on the states of neighbors, it is said to be indirect.

In Figure 3, the transition δ_{11} with respect to cells, c_1 and c_2 is **indirect local** and **non-local** respectively. Whereas transition δ_{21} with respect to cell, c_2 is **direct local**.

3.1 Conflict Resolution

When transition involves more than one cell, competition of resources will occur naturally. For instance (Figure 4), the current states of **Robot1**, **Robot2**, **Bin**, and **Iden_Reco** are **Pick_up**, **Requesting Parts**, **Parts Available**, and **IR Available** respectively. The state of **Bin** will transit from **Parts Available** to **Bin Occupied** driven by either the state of **Robot1** or **Robot2**. If it is driven by **Robot1**, no conflict will occur. However, **Robot2** will wait until parts become available. If it is driven by **Robot2**, a conflict will occur. Both **Robot1** and **Robot2** are at the **Pick_up** state and **Bin** is at **Bin Occupied** state (Figure 5). The conflict is *which robot will perform the action of picking up a part from the bin eventually*. An explicit conflict resolution scheme can be included in the

Figure 4: A Configuration from the Shen & Yan Model

model. This can be done easily by arbitrarily giving priority to cells. However, there are situations when assigning priority to resolve conflict will cause ambiguity in the operation of the process or even errors. For example (Figure 6), if **Robot2** were given the priority, it will pick up a part and transit to Sensor Inspection state. This will cause **Bin** to transit back to Parts Available. **Robot2** will continue its task. Now the problem occurs in the **Robot1** cell. There is no indication that whether **Robot1** did not pick up a part. It will transit to Sensor Inspection and an error will occur.

To solve this problem, we need to assign priority not only to cells but also to transitions. In this extended model, we allow transition to cause state change in more than one cell. Therefore, in Figure 7, high priority of “1” is given to transition δ_{21} (δ_{22}) which essentially

Figure 5: A Conflict Situation

Figure 6: An Error will occur

Figure 7: Priority assigned to Transitions

implies that **pick_up** has precedence over **Requesting Parts**. A higher priority of “2” is given to **Robot1** if both robots are in the **Requesting Parts** state. Transition δ_{12} (δ_{32}) in **Robot1** (**Robot2**) causes change of states in **Robot1** (**Robot2**) (from **Pick_up** to **Sensor_Inspection**) and **Bin** (from **Bin Occupied** to **Parts Available**) simultaneously. Figures 7, 8 and 9 give three consecutive configurations of the system. **Robot1** will have picked up the part and proceeded to **Sensor_Inspection**. **Robot2** will be waiting until parts become available.

Figure 8: Conflict resolved

Figure 9: Next time instant

3.2 Problem of Starvation

In any production system, low utilization of resources means ultimate loss in profit to the company. From the manufacturing systems point of view, low utilization of resources will cause over/under stock of certain parts and delay in production. Two phenomena (common in computer systems), namely starvation and deadlock², are the usual causes of low utilization of resources. We refer the phenomenon, *starvation*, when a resource is idle because of lack of components to work upon. In Figure 10, if the three transitions, δ_{1i} , δ_{2j} and δ_{3k} are not synchronized, starvation will occur. For example, if δ_{2j} took place before δ_{1i} and δ_{3k} , *Cell 1* and *Cell 3* will remain in the current state forever. The following two rules will ensure certain transitions are synchronized and thus prevent starvation.

Rule 1 *Any transition involving neighbours has to be resolved at the same time with any direct local transition in any of the neighbours concerned.*

Rule 2 *Any set of non-local, indirect transitions involving n cells in the model must be resolved simultaneously.*

Therefore, in Figure 10, transitions $\delta_{1i}, \delta_{2j}, \delta_{3k}$ must occur simultaneously. Otherwise, one of the cells will be idle and any production in the corresponding cell will be delayed.

4 Implementation

We have developed a prototype system to model FMS. The front-end provides a user-friendly interface for designing the specification while the back-end interprets the specification and serves as an executable specification language for FMS. Interfacing between

---

²The problem of deadlock is under investigation and will be reported in due course

Figure 10: Three transitions which are inter-dependent

the front-end and the back-end are done through demand-driven fashion. The overall architecture provides a platform for complex systems' specification, analysis and animation (Figure 11).

The Graphical User Interface, which is written in C with X-Window programming, provides a user-friendly interface for designing the specification of cellular automaton. As shown in Figure 12, user can design the graphical specification by selecting the appropriate buttons.

Each graphical specification can be compiled to a set of Constraint Narrowing Grammar (CNG) rules, a first-order rewrite rule language. The CNG rules generated from the graphical specification (Figure 12) are shown in appendix A. The resulting set of grammar rules can be directly executable.

The front-end is connected to a back-end CNG interpreter. We have adopted the algorithm in [2] to compile grammar rules to logic programs (an CNG interpreter). The resulting logic program serves as an executable specification for Flexible Manufacturing Systems. With appropriate extensions, it can be used as simulation and performance analysis tool.

Figure 11: Functional diagram of the VCAM

5 Conclusion

The initial model of FMS using cellular automata mentioned in [8] has been revised. Some currency problems of the model have been addressed and solved by extending the model. This extended model has been formally defined both mathematically and graphically. The graphical model can provide the precise overall picture of the system which makes it suitable for FMS design and illustration. This model is useful for modeling some important characteristics of FMS such as conflict resolution, dependency and starvation of resources. In order to facilitate the specification and animation of the model, a specification language tool has been implemented. Ongoing research is to design and develop appropriate extensions to the prototype system to be used as simulation and performance analysis tool.

Figure 12: Layout of the GUI

6 Acknowledgement

This research is supported by an RGC Competitive Earmarked Research Grant (project no. HKUST 184/93E), Hong Kong.

References

- [1] R. Baer and H. Martinez, *Automata and Biology*, Ann. Rev. Biophys. 3, 1974.

- [2] H.L. Chau and D.S. Parker, *Narrowing Grammar: Theory, Implementation, and Applications*, Journal of Logic Programming, Nov. 1992.
- [3] H.L. Chau and G.K. Chan, *A Visual Logical Specification of Flexible Manufacturing Systems*, Proceedings of the 6th International Conference on Software Engineering and Knowledge Engineering, pp. 438-445, 1994.
- [4] E.F. Codd, *Cellular Automata*, Academic Press, 1968.
- [5] A.A. Desrochers, *Modeling and Control of Automated Manufacturing Systems*, IEEE Computer Society Press, 1990.
- [6] T. Legendi, D. Parkinson, R. Vollmar, and G. Wolf, *Parallel Processing by Cellular Automata and Arrays*, 86' the III Inter. Workshop in Parallel Processing by Cellular Automata and Arrays, North-Holland, 1987.
- [7] H.C. Shen and W.P. Yan, *A Finite Automaton Model for an Autonomous Workcell*, 1990 Proceedings of the Pacific Conference on Manufacturing, pp109-116, Sydney and Melbourne, Australia, Dec 1990.
- [8] H.C. Shen and W.P. Yan, *Modeling Autonomous Assembly Systems and FMS using Cellular Automata*, The International Journal of Advanced Manufacturing Technology, Springer-Verlag London Ltd., 7:333-338, 1992.
- [9] J. Von Neumann, *Theory of automata: Construction, Reproduction, Homogeneity*, Part II of *Theory of Self-reproducing Automata*, A.W. Burks (ed.), University of Illinois Press, Urbana, Illinois, 1966.
- [10] S. Wolfram, *Theory and Application of Cellular Automata*, The World Scientific, 1986.

Appendix A

Constraint Narrowing Grammar (CNG) rules generated from the graphical specification :

```
init  $\Rightarrow$  concurrent([cell('robot1',nil,robot1),cell('robot2',nil,robot 2),cell('iden_reco',nil,iden_reco),
    cell('bin',nil,bin)]).

robot1  $\Rightarrow$  robot1_req_parts.
robot1_req_parts  $\Rightarrow$  [state(req_parts),in([source('bin',parts_avai l)]),robot1_pick_up.
robot1_pick_up  $\Rightarrow$  [state(pick_up),in([source('bin',pick_by_1)])] ,robot1_sensor_insp.
robot1_sensor_insp  $\Rightarrow$  [state(sensor_insp)],robot1_iden_reco_req .
robot1_sensor_insp  $\Rightarrow$  [state(sensor_insp)],robot1_assemble.
robot1_iden_reco_req  $\Rightarrow$  [state(iden_reco_req),in([source('iden_ reco',finish_ir1)])],robot1_assemble.
robot1_assemble  $\Rightarrow$  [state(assemble)],robot1_req_parts.

robot2  $\Rightarrow$  robot2_req_parts.
robot2_req_parts  $\Rightarrow$  [state(req_parts),in([source('bin',parts_avai l)]),robot2_pick_up.
robot2_pick_up  $\Rightarrow$  [state(pick_up),in([source('bin',pick_by_2)])] ,robot2_sensor_insp.
robot2_sensor_insp  $\Rightarrow$  [state(sensor_insp)],robot2_iden_reco_req .
robot2_sensor_insp  $\Rightarrow$  [state(sensor_insp)],robot2_assemble.
robot2_iden_reco_req  $\Rightarrow$  [state(iden_reco_req),in([source('iden_ reco',finish_ir2)])],robot2_assemble.
robot2_assemble  $\Rightarrow$  [state(assemble)],robot2_req_parts.

iden_reco  $\Rightarrow$  iden_reco_ir_avail.
iden_reco_ir_avail  $\Rightarrow$  [state(ir_avail),in([source('robot1',iden_ reco_req)])],iden_reco_iden_reco1.
iden_reco_ir_avail  $\Rightarrow$  [state(ir_avail),in([source('robot2',iden_ reco_req)])],iden_reco_iden_reco2.
iden_reco_iden_reco1  $\Rightarrow$  [state(iden_reco1)],iden_reco_finish_i r1.
iden_reco_iden_reco2  $\Rightarrow$  [state(iden_reco2)],iden_reco_finish_i r2.
iden_reco_finish_ir1  $\Rightarrow$  [state(finish_ir1)],iden_reco_ir_avail .
iden_reco_finish_ir2  $\Rightarrow$  [state(finish_ir2)],iden_reco_ir_avail .

bin  $\Rightarrow$  bin_parts_avail.
bin_parts_avail  $\Rightarrow$  [state(parts_avail),in([source('robot1',pick_u p)])],bin_pick_by_1.
bin_parts_avail  $\Rightarrow$  [state(parts_avail),in([source('robot2',pick_u p)])],bin_pick_by_2.
bin_pick_by_1  $\Rightarrow$  [state(pick_by_1)],bin_bin_occ.
bin_pick_by_2  $\Rightarrow$  [state(pick_by_2)],bin_bin_occ.
bin_bin_occ  $\Rightarrow$  [state(bin_occ),in([source('robot1',sensor_insp)])] ,bin_parts_avail.
```

```
bin_bin_occ ⇒ [state(bin_occ),in([source('robot2',sensor_insp)) ] ,bin_parts_avail.
```