Koh, N W, C Zielinski, M H Jr Ang and S Y Lim, "Matrix_based supervisory controller for transition_function specified robot controllers". *Proceedings of the Sixteenth CISM_IFToMM Symposium, Romansy 16, Robot Design, Dynamics and Control*, ed. Teresa Zielinska and Cezary Zielinski (2006): 229-236. Warsaw: Springer. (Romansy 16 Robot Design, Dynamics and Control, 21 _ 24 Jun 2006, WarsawUniversityofTechnology,Warsaw,Poland) (Part of EERSS project withWarsaw, 2004_2007).

Matrix-based Supervisory Controller of Transition-Function Specified Robot Controllers

Niak Wu Koh[*], Cezary Zieliński[†], Marcelo H. Ang, Jr.[*] and Ser Yong Lim [‡]

[*] National University of Singapore, Department of Mechanical Engineering
email: (nwkoh, mpeangh)@nus.edu.sg

[†] Warsaw University of Technology, Faculty of Electronics and Information Technology
email: C.Zielinski@ia.pw.edu.pl

[‡] Singapore Institute of Manufacturing Technology
email: sylim@SIMTech.a-star.edu.sg

Abstract. Robot control systems generally require a layer that manages, in a modular manner, the hardware or low level control and a supervisory layer which allows task specification for the execution of a task. This paper proposes the transition-function specified controllers as a lower layer, with an improved matrix-based supervisory controller for the high level operations.

1 Introduction

With a growing popularity in discrete event systems, its usage has been largely favoured by the robotics and manufacturing community especially from the stance of task planning and modelling (Morandin and Kato, 2005; Fourquet et al., 2005). Task planning describes the ability to decompose a task into a set of primitives. Recomposing this primitive set into a cognitively logical sequence, which when executed, enables the task to be carried out autonomously or with user intervention as and when needed.

Whether a primitive has completed its execution is dependent on the current and terminal state of the system. Therefore, a framework that caters to the development of these primitives and the transmission of the system's sensor information is mandatory. Control of a system essentially boils down to the processing of information about the current state of the system and its environment enabling a defined task to be executed.

The remainder of this article is organized as follows: Section 2 introduces the matrix-based controller. Section 3 provides a brief outlook into the transition-function based approach while Section 4 proposes the use of the matrix model as a supervisory controller. Section 5 concludes.

2 Matrix-based Discrete Event Controller

With the use of manufacturing engineering concepts, a powerful rule-based matrix model was developed by Tacconi and Lewis (1997) which has only been implemented to flexible manufacturing systems (see Boghan et al., 2002; Mireles and Lewis, 2001). The pursuit of

this research is to employ the discrete event controller to showcase its potential as a tool for the execution of complex robotic tasks. Preliminary work regarding the development and implementation of the matrix model from the modified to its improved state can be viewed in Koh et al. (2005a) and Koh et al. (2005b) respectively.

A task consists of a sequence of jobs and its corresponding resources which, via the use of input signals and a set of conditions, triggers the execution of each job. The entire task is complete once an output signal is present.

With the matrix model, assembly/job sequencing, addition of resources, analyses for deadlock and its avoidance and a dispatching design can be carried out allowing a thorough analysis with a convenient solution to the simulation of the system in the form of a Petri Net (Murata, 1989). The improved matrix discrete event model is described in the subsequent sections.

2.1 Discrete Event Model State Equation

The task can be described by:

$$\begin{gathered} \bar{x} = F_v \otimes \bar{v}_c \oplus F_r \otimes \bar{r}_c \oplus F_u \otimes \bar{u} \oplus F_D \otimes \bar{u}_D \\ \rightarrow \bar{x}_i = F_v(i,j) \otimes \bar{v}_{c_j} \oplus F_r(i,j) \otimes \bar{r}_{c_j} \oplus F_u(i,j) \otimes \bar{u}_j \oplus F_D(i,j) \otimes \bar{u}_{D_j} \end{gathered} \tag{2.1}$$

The condition (x) represents the state of the system and the equations shows how it evolves over time.

Table 1. Variable definitions for Equation (2.1)

i, j	[Row, Column]
F_v	Job sequencing matrix
F_r	Resource requirements matrix
F_u	Parts input matrix
F_D	Dispatching matrix

x	Condition
v	Job vector
r	Resource vector
u	Input signal
u_D	Dispatch control vector

As the equations used in the matrix model are logical equations, standard matrix multiplication and addition are replaced by AND/OR algebra and all vectors and matrices are binary. $\otimes$ represents an AND operation and $\oplus$ an OR operation. The over bar is a logical negation and is defined as follows: For any component $a(i)$ of a natural number vector **a**

$$\bar{a} = 0 \quad if \quad a(i) > 0 \qquad \bar{a} = 1 \quad if \quad a(i) \leq 0$$

Each of the matrices are explained as follows:

- F_v determines which relevant job should be completed before condition x is satisfied. When $v_c = 1$, a job is said to be complete.
- F_r determines which relevant resources should be present before condition x is satisfied. When $r_c = 1$, a resource is said to be currently available.

- F_u determines which relevant input signal should be present before condition x is satisfied. When $u = 1$, an input signal is said to be present.
- F_D determines which relevant dispatch signal should be present before condition x is satisfied. When $u_D = 1$, a dispatch signal is said to be present. This matrix is used to determine the priority of the operations when resources are shared.

2.2 Job Start Equation

The start of a job is indicated by the following equation:

$$\begin{gathered} \bar{v}_s = S_v \otimes \bar{x} \oplus U \otimes \bar{v}_c \\ \rightarrow \bar{v}_{s_i} = S_v(i,j) \otimes \bar{x}_j \oplus U(i,j) \otimes \bar{v}_{c_j} \qquad v_{s_i} \neq v_{c_j} \end{gathered} \tag{2.2}$$

where S_v is a rectangular job start matrix and U is any user-defined matrix of n x m dimensions (n can equal m). n will assume the number of elements in the $\bar{v}_s$ column vector. Job i starts when $v_{s_i} = 1$. Equation (2.2) can be read as follows: Job i will start iff the relevant conditions (determined by S_v) are satisfied and the relevant job(s) is/are complete. This general representation can be used for a concurrent and dependent operation.

For a sequential operation

$$\bar{v}_s = S_v \otimes \bar{x} \oplus I \otimes \begin{bmatrix} 0 \\ \bar{v}_c \end{bmatrix} \tag{2.3}$$

where I is an n x n identity matrix with n assuming the number of elements in the $\bar{v}_s$ column vector. It is worth noting that the presence of the $\bar{v}_c$ element in the latter part of the equation serves as an added validity check since the status of $\bar{v}_c$ is already present in $\bar{x}$.

By decomposing a complex task into a series of elemental sequences, task primitives, used to represent each subtask, can be assigned to the job start vector v_s.

2.3 Resource Release Equation

The equation indicating the release of a resource is:

$$\begin{gathered} \bar{r}_s = S_r \otimes \bar{x} \oplus U \otimes \bar{r}_c \\ \rightarrow \bar{r}_{s_i} = S_r(i,j) \otimes \bar{x}_j \oplus U(i,j) \otimes \bar{r}_{c_j} \end{gathered} \tag{2.4}$$

where S_r is a rectangular resource release matrix and U is any user-defined matrix of n x m dimensions (n can equal m). n will assume the number of elements in the $\bar{r}_s$ column vector. Resource i is released when $r_{s_i} = 1$. Equation (2.4) can be read as follows: Resource i can only be released iff the relevant conditions (determined by S_r) are satisfied and the relevant user-defined resource(s) is/are idle.

For a sequential operation

$$\bar{r}_s = S_r \otimes \bar{x} \oplus I \otimes \bar{r}_c \tag{2.5}$$

where I is an n x n identity matrix with n assuming the number of elements in the $\bar{r}_s$ column vector. It is worth noting that the presence of the $\bar{r}_c$ element in the latter part of the equation serves as an added validity check since the status of $\bar{r}_c$ is already present in $\bar{x}$.

2.4 Process Output Equation

S_y determines the set of conditions that need to be satisfied before a product is outputted:

$$\bar{y} = S_y \otimes \bar{x} \rightarrow \bar{y}_i = S_y(i,j) \otimes \bar{x}_j \tag{2.6}$$

A theoretical insight to the improved matrix-based supervisory controller is discussed in Koh et al. (2005c). Figure 1 depicts the matrix model. The S matrices determine which set of conditions must be satisfied to release the resources required by the starting of a job. Once a job is complete, the F matrices is used to obtain the next set of conditions, based on the system's current state, which will be used by the S matrices.

The matrix-based supervisory controller allows an effective and simple online control by applying a set of discrete event control signals. Together with the Petri net marking transition equation (Murata, 1989), the matrix formulation provides a complete dynamical description that can be used for analysis and computer simulation (Koh et al., 2005b).

3 Behavioural Control Case in a Nutshell

Task planning begins with the decomposition of a complex task into a sequential series of elemental subtasks/jobs/primitives. From the point of view of clarity of a task description that is to be executed by a system, it is useful to group the steps of the commanded evolution of the control subsystem state into sequences which are termed as primitive behaviours (Zielinski, 2005). Once a decomposed cognitive plan of a task is available, Equation (2.2) is used where v_s would represent the primitives.

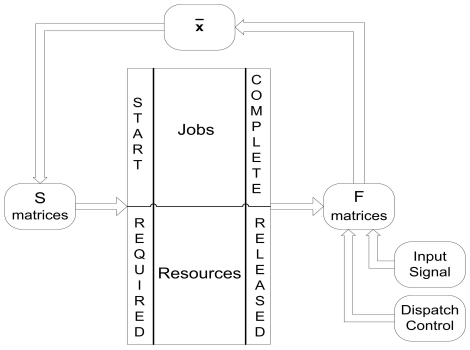

Figure 1. Depiction of the Matrix-based Discrete Event Controller.

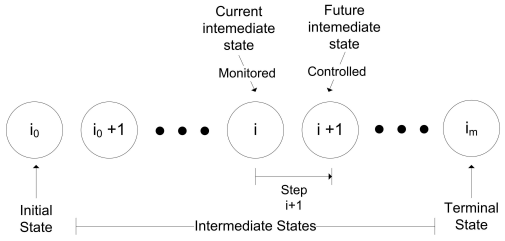

Figure 2. Evolution of the system state during the execution of an instruction/job.

3.1 Behaviour of an Agent

A behaviour can be defined as a sequence of total states

$$ {}^{q}b_j^i = {}^{q}b_j = \{c_j^{i+1}, c_j^{i+2}, ..., c_j^{i+n_s}\} \tag{3.1} $$

where n_s is the number of steps in a behaviour and q denotes a numeric identifier of this reaction, i denotes the current instant and the next considered instant is denoted by $i+1$. Each sequence of states $c_j^{i+1}, c_j^{i+2}, ..., c_j^{i+n_s}$ is generated by one of the functions ${}^{m}f_{c_j}$, i.e., the function defining the primitive behaviour. n_f partial functions are defined as

$$ {}_{y}c_j^{i+1} = {}^{m}f_{c_j}({}_{x}c_j^i) \qquad m = 1, ..., n_f \tag{3.2} $$

whereby the control subsystem uses

$$ {}_{x}c_j^i = \{c_{c_j}^i, {}_{x}c_{e_j}^i, {}_{x}c_{V_j}^i, {}_{x}c_{T_j}^i\} \tag{3.3} $$

to produce

$$ {}_{y}c_j^{i+1} = \{c_{c_j}^{i+1}, {}_{y}c_{e_j}^{i+1}, {}_{y}c_{V_j}^{i+1}, {}_{y}c_{T_j}^{i+1}\} \tag{3.4} $$

and hence the transition functions

$$ \begin{array}{l} c_{c_j}^{i+1} = f_{c_{c_j}}(c_{c_j}^i, {}_{x}c_{e_j}^i, {}_{x}c_{V_j}^i, {}_{x}c_{T_j}^i) \\ {}_{y}c_{e_j}^{i+1} = f_{c_{e_j}}(c_{c_j}^i, {}_{x}c_{e_j}^i, {}_{x}c_{V_j}^i, {}_{x}c_{T_j}^i) \\ {}_{y}c_{V_j}^{i+1} = f_{c_{V_j}}(c_{c_j}^i, {}_{x}c_{e_j}^i, {}_{x}c_{V_j}^i, {}_{x}c_{T_j}^i) \\ {}_{y}c_{T_j}^{i+1} = f_{c_{T_j}}(c_{c_j}^i, {}_{x}c_{e_j}^i, {}_{x}c_{V_j}^i, {}_{x}c_{T_j}^i) \end{array} \tag{3.5} $$

c_j is the state of the control subsystem of an agent
e_j is the state of the effector of the agent
V_j is the bundle of virtual sensor readings and
T_j is the information from/to the other agents

The state of an agent is thus defined as

$$ s_j = \{c_j, e_j, V_j, T_j\} \tag{3.6} $$

Each program instruction causes a certain change in the state of a system. If we were to associate with an instruction a state-transition pair: *initial state* and *terminal state*, a finite number of discrete event states can be defined using the mapping *sem* as

$$ sem : II \mapsto [S \rightarrow S^*] \tag{3.7} $$

where ii is an instruction and II the set of instructions, $ii \, \epsilon \, II$.

This formal approach to the behavioural aspect of a robot programming framework was developed by Zielinski (2005)(see Zielinski, 1995 for more details).

In the case of a purely reactive system, the choice of the function ${}^{m}f_{c_j}$ is based on testing predicates ${}^{q}p_{c_j}$, $q = 1, .., n_p$, which take as arguments only the components of ${}_{x}c_{V_j}^i$. In pseudo-code, it can be expressed as

$$ if \ {}^{q}p_{c_j}({}_{x}c_{V_j}^i) \ then \ {}_{y}c_j^{i+1} := {}^{m}f_{c_j}({}_{x}c_j^i) \tag{3.8} $$

Pseudo-code (3.8) represents a single-step behaviour, i.e., $n_s = 1$ (refer to Equation (3.1) for the definition of n_s). For the case of a multi-step behaviour, the pseudo-code is expressed as

$$if\ {}^{q}p_{c_j}({}_{x}c_j^i)\ then\ {}^{q}b_j({}_{x}c_j^i) \tag{3.9}$$

The control program is composed of an endless loop containing a sequence of instructions of form (3.9). Each iteration of the loop contains several control steps i. The required computations, i.e., computation of ${}_{y}c_j^{i+\epsilon}$, $\epsilon = 1, .., n_s$, and the execution of behaviours, i.e., transmissions $e_j \mapsto {}_{x}c_{e_j}^i$, $V_j \mapsto {}_{x}c_{V_j}^i$, $c_{T_j} \mapsto {}_{x}c_{T_j}^i$, are bundled together within ${}^{q}b_{c_j}({}_{x}c_j^i)$, $q = 1, .., n_p$.

4 Sensor Utilization

If notation (3.7) is assumed, the execution of an instruction begins in an initial state, ends in a terminal state and traverses a sequence of intermediate states. Robots are generally controlled by digital computers so the execution of each instruction is subdivided into steps. Each step results in a change of the system state from one intermediate step to the next.

In each intermediate (or current) state, the state of a system can be measured or monitored by sensors. However, the future intermediate states can only be influenced, i.e., controlled. An initial state can be treated as a current intermediate state at the beginning of an instruction execution. A terminal state is the current intermediate state in which the execution of the instruction terminates. As the initial and terminal states are special cases of current intermediate states, both of them can only be monitored (see Figure 2).

The monitoring of the system state is performed by receptors. Raw data obtained from them cannot be utilized directly to monitor or control a system. It has to be transformed into a useful form by data aggregation. In consequence, a virtual sensor reading, V, is obtained.

5 Matrix Model as the Supervisory Controller

Consider a simple task (see Figure 3) where an end-effector moves to two different coordinates (jobs v_{s_1} and v_{s_2}) in cartesian space. With reference to Equation (2.3),

$$\begin{bmatrix} \bar{v}_{s_1} \\ \bar{v}_{s_2} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} \bar{x}_1 \\ \bar{x}_2 \\ \bar{x}_3 \end{bmatrix} \oplus \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ \bar{v}_{c_1} \end{bmatrix} \tag{5.1}$$

$$\begin{aligned} \bar{v}_{s_1} = \bar{x}_1 \ &\rightarrow\ v_{s_1} = x_1 \\ \bar{v}_{s_2} = \bar{x}_2 \oplus \bar{v}_{c_1} \ &\rightarrow\ v_{s_2} = x_2 \otimes v_{c_1} \end{aligned}$$

which is read as follows: Job 1 can start iff condition x_1 is satisfied; Job 2 can start iff condition x_2 is satisfied and job 1 is complete.

In concordance with Figure 2, a *Move* instruction can be depicted as shown in Figure 4. With the initial and terminal states (monitored), and by employing the formal

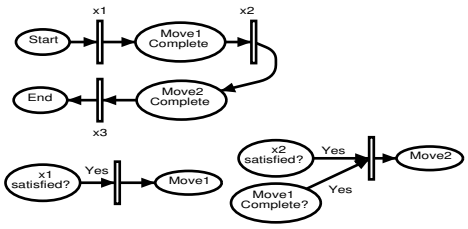

Figure 3. Petri Net representation of a simple move task. Circles/ellipses represent the places and rectangles are the transitions.

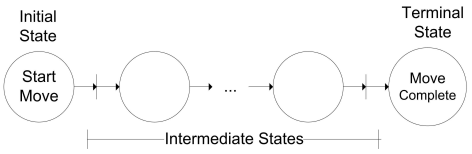

Figure 4. Petri Net representation of a general move instruction.

framework discussed above, the current status of a job or an action can be determined since the aggregated virtual sensor readings, V, are available.

When certain conditions are satisfied and a job can be executed, the supervisor sends a command to the control level (see Figure 5). Once a job is complete, the transition-function specified controller will pass a token, through transmission buffers, T, to the higher level supervisory controller signifying completion of a particular action. Therefore, the supervisory controller *sees* only the tokens providing transparency of the lower level.

Condition, x, quantifies the current state of the system in terms of the availability of resources and the completion of jobs via sensor utilization. Since the state of the system changes with time, x is thus dynamic. Depending on the virtual sensor readings, any one of the vector of conditions can be satisfied which in turn will trigger the start of a job or the release of a resource.

By manipulating the matrix entities in Equation (5.1), the sequence of the jobs and its dependents can be altered without the need to re-structure a program for the execution of a separate task. Viewing each job as a task primitive, the decision to opt for a completely different task is possible simply by choosing the required primitive from a predefined library after task decomposition.

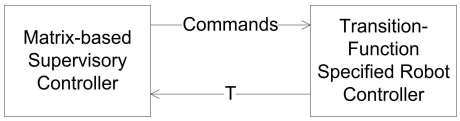

Figure 5. Matrix-based Supervisory Controller with the Transition-Function Specified Controllers. T is the transmission buffer.

6 Conclusion

Robotic tasks in general require a great deal of robustness since the system might encounter some unforeseen problem preventing the task from completion. Information regarding the current and terminal state of a primitive behaviour is essential to the su-

pervisory controller for the execution of a task. By employing the transition-function based approach at the lower level, this necessity is realized.

This framework adopts a transition function-based formalism which introduces rigor into the design and implementation of a single or multi-agent system. By decomposing a large system into modular components, the design and implementation solely by providing the relevant code for the specified functions would suffice for an operational system. Through their public interfaces, these components provide only the data that is necessary for the computation of the control of an agent which is utilized by the transition functions resident in the control subsystem to compute the next state of this subsystem.

Bibliography

S. Boghan, F. L. Lewis, Z. Kovacic, A. Grel, and M. Stajdohar. An implementation of the matrix-based supervisory controller of flexible manufacturing systems. In *IEEE Trans. On Control Systems Technology*, volume 10 of *5*, pages 709–716, September 2002.

J-Y. Fourquet, V. Padois, P. Chiron, and A. Mauratille. Reactive behavior and dynamic sequencing for nonholonomic mobile manipulators. In *IEEE Int. Conference on Computational Intelligence, Robotics and Autonomous Systems*, 13-16 December 2005.

N. W. Koh, M. H. Ang Jr, and S. Y. Lim. Implementation of a matrix-based discrete event controller for robotic tasks. In *IEEE Asian Conf. for Industrial Automation and Robotics*, 11-13 May 2005a.

N. W. Koh, M. H. Ang Jr, and S. Y. Lim. Robotic tasks employing an improved matrix-based discrete event controller. In *Int. Symp. on Collaborative Research in Applied Science*, 7-9 October 2005b.

N. W. Koh, M. H. Ang Jr, and S. Y. Lim. A theoretical insight to the improved matrix-based supervisory controller. In *IEEE Int. Conf. on Computational Intelligence, Robotics and Autonomous Systems*, 13-16 December 2005c.

Jr J. Mireles and F. L. Lewis. Intelligent material handling: Development and implementation of a matrix-based discrete-event controller. In *IEEE Trans. on Industrial Electronics*, volume 48 of *6*, pages 1087–1097, December 2001.

Jr O. Morandin and Edilson R.R. Kato. Virtual petri nets as a modular modeling method for planning and control tasks of fms. In *Int. J. Computer Integrated Manufacturing*, volume 18, pages 100–106. Federal University of Sao Carlos, Taylor and Francis Group, March-May 2005.

T. Murata. Petri nets: Properties, analysis and applications. In *Proceedings of IEEE*, volume 77, pages 542–580, April 1989.

D. A. Tacconi and F. L. Lewis. A new matrix model for discrete event systems: Application to simulation. In *IEEE Control Systems*, volume 17 of *5*, pages 62–71, October 1997.

C. Zielinski. Robot programming methods. DSc thesis, Politechnika Warszawska, 1995.

C. Zielinski. Formal approach to the design of robot programming frameworks: The behavioural control case. In *Bulletin of the Polish Academy of Sciences*, volume 53, pages 1–11, 2005.