2007 IEEE International Conference on
Robotics and Automation
Roma, Italy, 10-14 April 2007

Job-agents: How to coordinate them?

Niak Wu Koh, Cezary Zieliński, Marcelo H. Ang, Jr. and Ser Yong Lim

***Abstract*— With our proposed decomposition into layers, a generic framework leading to the reuse of previously produced software and the extraction of useful portions can be achieved. The presented transition function based formalism can be applied to specifying programming frameworks for robot controllers executing very diverse tasks. Formalization introduces rigor into the discussion of the structure of embodied agent controllers. The paper deals with systems consisting of multiple agents executing jobs by: influencing the environment through effectors, gathering information from the environment through sensors and communicating with the other agents through communication channels. A paradigm shift is proposed: from building systems executing jobs, to agents acquiring resources (effectors and receptors) so that they can execute the jobs assigned to them. A supervisory controller coordinates the adequate sequencing of jobs and resolves contentions.**

I. INTRODUCTION

Robots will be ubiquitous with their complexity masked behind a user interface [2][11]. The underlying engine of such a system takes root from a programming architecture stemming from the experience (tacit knowledge) of a roboticist. However, if the previously produced software does not have a structure facilitating its reuse, it is usually very difficult to extract the useful portions. Moreover, any modification or extension of the old software might be hindered by its inadequate structure. This is especially true for robot control software. It is relatively easy to produce code for a specific device and a specific task, but when those change, it is sometimes easier to start coding from scratch than try to reuse the old pieces [9]. The motivation of our work thus lies in the formalization of a generic robot programming framework which utilizes a matrix-based supervisory controller as its engine. Issues that will be tackled in this paper, however, address a programming methodology taking into account that a task requires resources for its execution. A coexisting problem handling the matter of resources is evident in grid computing [3][4].

The remainder of the article is organized as follows: Section II proposes a paradigm shift in the perception of a job. Section III describes a method for job coordination. Both of these sections present the transition function based formalism. Details of the coordination method are discussed in Section IV with the issues of exception handling in Section V. Section VI briefly discusses an application example while Section VII concludes.

N.W. Koh and M.H. Ang, Jr. are with the Department of Mechanical Engineering, National University of Singapore, Singapore {nwkoh,mpeangh}@nus.edu.sg

C. Zieliński is with the Faculty of Electronics and Information Technology, Warsaw University of Technology, Poland C.Zielinski@ia.pw.edu.pl

S.Y. Lim is with the Singapore Institute of Manufacturing Technology, Singapore sylim@SIMTech.a-star.edu.sg

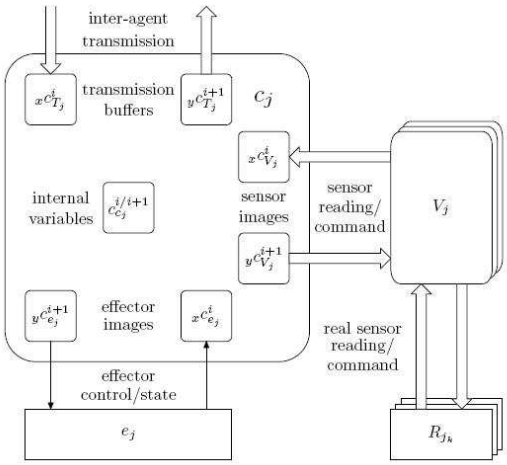

Fig. 1. An embodied agent – an agent that acquired the necessary resources (effectors and receptors) for the execution of its job

II. JOBS AS AGENTS

In a strictly conventional sense, a resource, which needs to be controlled, is assigned to carry out a job. Departing from the orthodoxies, a paradigm shift has been proposed such that instead of assigning a job to a resource, the job now acquires resources to execute *itself*. The controller is now associated with the job which embodies itself on a resource(s). This suggests that the job is now a virtual agent.

A system consisting of n_a job-agents (Fig. 1) is considered. The state of the control system of the agent a_j, $j = 1, .., n_a$, at time instant i, is:

$$c_j^i = \langle c_{c_j}^i,\ {}_xc_{e_j}^i,\ {}_xc_{V_j}^i,\ {}_xc_{T_j}^i,\ {}_yc_{e_j}^i,\ {}_yc_{V_j}^i,\ {}_yc_{T_j}^i \rangle, \qquad (1)$$

where

- ${}_xc_{e_j}$ – input from the effectors,
- ${}_xc_{V_j}$ – input from the virtual sensors (aggregated readings from receptors – hardware sensors $R_{j_k}, k = 1, \ldots, n_R$),
- ${}_xc_{T_j}$ – input from the inter-agent transmission (information obtained from other agents),
- ${}_yc_{e_j}$ – control of the effectors,
- ${}_yc_{V_j}$ – commands to the virtual sensors,
- ${}_yc_{T_j}$ – output to the inter-agent transmission (information transmitted to other agents),
- c_{c_j} – all the other relevant variables taking part in data processing within the agent's control subsystem.

The agent uses the input (subscript x) and output (subscript y) buffers to communicate with its resources (effectors and receptors) and the other agents – in this paper we consider only the communication with the supervisor.

The agent can be in any of the following four general states W – Waiting (actively idle), R – Running (executing itself), F – Finished (executed itself) and E – Error (unsuccessfully finished). Within the structure c_{c_j}, a component that holds the current general state of the agent is singled out – let that component be ${}^s c_{c_j}$.

$$ {}^s c_{c_j} \in \{W, R, F, E\} \tag{2} $$

The agent (job) a_j acquires its effectors e_j and its virtual sensors V_j (and thus its receptors R_j) in the state W – gradually embodying itself. Once all the resources have been acquired it can run. Upon completion of the job (state F) or in the event of an error that cannot be remedied (state E) the resources are returned and thus the agent disembodies itself. However, throughout the lifetime of the agent, i.e., in all of its general states, it retains its ability to communicate with other agents through c_{T_j}.

If i denotes the current instant, the next considered instant is denoted by $i+1$. The agent uses

$$ {}_x c_j^i = \langle c_{c_j}^i, {}_x c_{e_j}^i, {}_x c_{V_j}^i, {}_x c_{T_j}^i \rangle, \tag{3} $$

to produce

$$ {}_y c_j^{i+1} = \langle c_{c_j}^{i+1}, {}_y c_{e_j}^{i+1}, {}_y c_{V_j}^{i+1}, {}_y c_{T_j}^{i+1} \rangle, \tag{4} $$

and hence the transition functions are defined as:

$$ \begin{array}{l} c_{c_j}^{i+1} = {}^m f_{c_{c_j}}(c_{c_j}^i, {}_x c_{e_j}^i, {}_x c_{V_j}^i, {}_x c_{T_j}^i) \\ {}_y c_{e_j}^{i+1} = {}^m f_{c_{e_j}}(c_{c_j}^i, {}_x c_{e_j}^i, {}_x c_{V_j}^i, {}_x c_{T_j}^i) \\ {}_y c_{V_j}^{i+1} = {}^m f_{c_{V_j}}(c_{c_j}^i, {}_x c_{e_j}^i, {}_x c_{V_j}^i, {}_x c_{T_j}^i) \\ {}_y c_{T_j}^{i+1} = {}^m f_{c_{T_j}}(c_{c_j}^i, {}_x c_{e_j}^i, {}_x c_{V_j}^i, {}_x c_{T_j}^i) \end{array} \tag{5} $$

or more compactly,

$$ {}_y c_j^{i+1} = {}^m f_{c_j}({}_x c_j^i), \tag{6} $$

where ${}^m f_{c_j}$ are the functions defining the primitive behaviour of the agent within each general state (e.g., W, R). As within each state many such functions may be used, hence the superscript m, $m = 1, \ldots, n_m$, where n_m is the number of such functions (the problem of switching between those functions is dealt with in [13][14]). Depending on the information obtained, conditions embedded within the agent will trigger the changing of those states (e.g., W to R, F to W) (Fig. 2). An elaborate primitive behaviour can be realized by the composition of transition functions.

While the job-agent is in the waiting state (and hence has not procured the resources), the control subsystem uses only

$$ {}_x c_j^i = \langle c_{c_j}^i, \bullet, \bullet, {}_x c_{T_j}^i \rangle, \tag{7} $$

to produce

$$ {}_y c_j^{i+1} = \langle c_{c_j}^{i+1}, \bullet, \bullet, {}_y c_{T_j}^{i+1} \rangle, \tag{8} $$

where $\bullet$ represents a missing (or illegal) argument. In this case the job-agent leads a virtual life in which it can do only computations and talk to other agents.

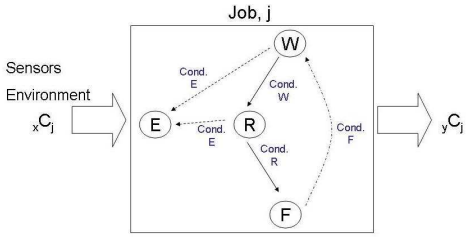

Fig. 2. Job-agent

III. Coordination

Given a situation with a number of job-agents, it is only natural that each agent will strive to procure a resource to execute itself. This competition escalates when agents that require the same resources are present. Who (or what) decides which agent gets to procure first? Our proposal is to produce a high level job coordinator a_0 employing a supervisory matrix-based approach (Section IV).

A. High Level Job Coordinator (HLJC)

The system now assumes a star topography with the supervisor a_0 at the center and the job-agents a_j surrounding it. The supervisor HLJC, like a job, is a virtual agent a_0 that senses the environment through its virtual sensors, whereby the information obtained will trigger a set of conditions which in turn will start the task. The relevant job-agents a_j are then alerted via the transmission buffers. The agent a_0 is virtual, because it does not acquire effectors – the only resources that it might require are the receptors, so it might lead a disembodied life, hence its virtuality. However it can be in any of the general states that the other agents can be in. The life of the system is initiated by first creating the agent a_0, so if the resources are at hand it is sure to acquire them (other agents do not yet exist). However, if the necessary resources are not available the agent a_0 goes into an error state E and the task cannot be accomplished.

The control system of the agent a_0, in its running state R, uses

$$ {}_x c_0^i = \langle c_{c_0}^i, \bullet, {}_x c_{V_0}^i, {}_x c_{T_0}^i \rangle, \tag{9} $$

to produce

$$ {}_y c_0^{i+1} = \langle c_{c_0}^{i+1}, \bullet, {}_y c_{V_0}^{i+1}, {}_y c_{T_0}^{i+1} \rangle, \tag{10} $$

where the components ${}_x c_{V_0}$ and ${}_x c_{T_0}$ provide the information about:

$\mathbf{r_c}$ – resource availability,

$\mathbf{a_c}$ – completed job-agents,

$\mathbf{u}$ – job input signals and

$\mathbf{u_D}$ – priority of a job.

The transition function governing the execution of the task is:

$$ {}_y c_0^{i+1} = f_{c_0}({}_x c_0^i), \tag{11} $$

Depending on the information obtained, conditions embedded within the function f_{c_0} will trigger the changing of the agents' general states (e.g., W to R, F to E). The structure

of the function (11) is governed by the equations defined in Section IV, i.e., (12)–(17).

IV. Matrix-based Approach

This matrix model was first introduced in [10]; applied to manufacturing systems [1] and more recently to wireless sensor networks [5]. Our development of the matrix model however, is novel to that of the initial idea with its introduction as an aid to robot programming [6][7].

The task can be described by:

$$\bar{\mathbf{x}} = (\Sigma^{\oplus}\mathbf{Q_a})\otimes\bar{\mathbf{a}}_c\oplus(\Sigma^{\oplus}\mathbf{Q_r})\otimes\bar{\mathbf{r}}_c\oplus\mathbf{Q_u}\otimes\bar{\mathbf{u}}\oplus\mathbf{Q_D}\otimes\bar{\mathbf{u}}_\mathbf{D} \quad (12)$$

Table I shows the variable definitions for (12). The condition ($\mathbf{x}$) represents the state of the task and the equations show how it evolves over time. 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, $\oplus$ an OR operation and $\Sigma^{\oplus}$ a logical OR summation. The over bar is a logical negation and is defined as follows: For any component z_k of a natural number vector $\mathbf{z}$

$$\bar{z}_k = 0 \quad if \quad z_k > 0, \qquad \bar{z}_k = 1 \quad if \quad z_k \leq 0$$

Each of the matrices are explained as follows:

- $\mathbf{Q_a}$ determines which relevant job-agent should be completed before condition x is satisfied. When $a_{j_c} = 1$, a job-agent is said to be complete.
- $\mathbf{Q_r}$ determines which relevant resources should be present before condition x is satisfied. When $r_{j_c} = 1$, a resource is said to be currently available.
- $\mathbf{Q_u}$ determines which relevant input signal should be present before condition x is satisfied. When $u_j = 1$, an input signal is said to be present.
- $\mathbf{Q_D}$ determines which relevant dispatch signal should be present before condition x is satisfied. When $u_{D_j} = 1$, a dispatch signal is said to be present. This matrix is used to determine the priority of the operations when resources are shared.

TABLE I
VARIABLE DEFINITIONS FOR (12)

Variables	Dimensions	Definitions
$\mathbf{x}$	n_x x 1	n_x number of conditions
$\mathbf{a}$	n_a x 1	n_a number of job-agents
$\mathbf{r}$	n_r x 1	n_r number of resources
$\mathbf{u}$	n_u x 1	n_u number of input signals
$\mathbf{u_D}$	n_{u_D} x 1	n_{u_D} number of dispatch controls
$\mathbf{Q_a}$	n_x x n_a	Job-agent sequencing matrix
$\mathbf{Q_r}$	n_x x n_r	Resource requirements matrix
$\mathbf{Q_u}$	n_x x n_u	Parts input matrix
$\mathbf{Q_D}$	n_x x n_{u_D}	Dispatching matrix

The state of each condition is dependent on the input received from the environment in terms of job-agent completion, resource availability, input signals and dispatch control. Upon successful condition satisfaction, a job and a resource will be triggered to start (Job-Agent Start Equation) and released (Resource Release Equation) respectively. Catering to the job-agent centric nature, two major components in (12) are briefly discussed in IV-A and IV-B.

A. Resources

With the manifestation of the job-agent, a method describing the selection of resource procurement in (12) is:

$$\bar{\mathbf{x}} = (\Sigma^{\oplus}\mathbf{Q_r})\otimes\bar{\mathbf{r}}_c$$

$$\rightarrow \bar{\mathbf{x}} = (\mathbf{Q_{r_1}}\otimes\bar{\mathbf{r}}_c)\oplus(\mathbf{Q_{r_2}}\otimes\bar{\mathbf{r}}_c)\oplus ... \oplus(\mathbf{Q_{r_n}}\otimes\bar{\mathbf{r}}_c), \quad (13)$$

where $\mathbf{Q_{r_k}}, k = 1, .., n$ are individual matrices of size n_x x n_r (Table I) that allow the assignment of a combination of n possible resources that *can* be acquired by a job-agent. In the event where multiple resources are available at the same instant, the job-agent decides (either independently or via the HLJC) which to procure.

B. Completed Job-Agents

The state equation (12) is also dependent on a vector of completed job-agents $\mathbf{a}$:

$$\bar{\mathbf{x}} = (\Sigma^{\oplus}\mathbf{Q_a})\otimes\bar{\mathbf{a}}_c$$

$$\rightarrow \bar{\mathbf{x}} = (\mathbf{Q_{a_1}}\otimes\bar{\mathbf{a}}_c)\oplus(\mathbf{Q_{a_2}}\otimes\bar{\mathbf{a}}_c)\oplus ... \oplus(\mathbf{Q_{a_n}}\otimes\bar{\mathbf{a}}_c), \quad (14)$$

where $\mathbf{Q_{a_k}}, k = 1, .., n$ are individual matrices of size n_x x n_a (Table I) that allow the assignment of n possible job-agents that *can* be completed before a condition is satisfied.

C. Job-Agent Start Equation

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

$$\bar{\mathbf{a}}_s = \mathbf{S_a}\otimes\bar{\mathbf{x}}\oplus\mathbf{U_a}\otimes\bar{\mathbf{a}}_c, \quad (15)$$

where $\mathbf{S_a}$ is a job-agent start matrix (n_a x n_x dimensions) and $\mathbf{U_a}$ is a job-agent dependency matrix (n_a x n_a dimensions. Job-agent $\mathbf{a_j}$, $j = 1, .., n_a$ starts when $a_{j_s} = 1$. Equation (15) can be read as follows: Job-agent $\mathbf{a_j}$ will start iff the relevant conditions (determined by $\mathbf{S_a}$) are satisfied and the job-agent dependencies (determined by $\mathbf{U_a}$) is/are complete. This general representation can be used for concurrent and dependent operations and is bridged by $\mathbf{U_a}$. From a broader perspective, (15) provides a means to

synchronize local and external subsystems in terms of job-agents.

D. Resource Release Equation

Before a job-agent can procure a resource, a resource first has to be released. The equation indicating the release of a resource is:

$$\bar{\mathbf{r}}_s = \mathbf{S_r} \otimes \bar{\mathbf{x}} \oplus \mathbf{U_r} \otimes \bar{\mathbf{r}}_c, \tag{16}$$

where $\mathbf{S_r}$ is a resource release matrix (n_r x n_x dimensions) and $\mathbf{U_r}$ is a resource dependency matrix (n_r x n_r dimensions). Resource $\mathbf{r_j}$, $j = 1, .., n_r$ is released when $r_{j_s} = 1$. Equation (16) can be read as follows: Resource $\mathbf{r_j}$ can only be released iff the relevant conditions (determined by $\mathbf{S_r}$) are satisfied and the resource dependencies (determined by $\mathbf{U_r}$) is/are currently available. Similarly, (16) provides a means to synchronize local and external subsystems in terms of resources.

E. Task Output Equation

$\mathbf{S_y}$ (n_T x n_x dimensions where n_T is the number of tasks) determines the set of conditions that need to be satisfied before the task ends:

$$\bar{\mathbf{y}} = \mathbf{S_y} \otimes \bar{\mathbf{x}} \tag{17}$$

Fig. 3 depicts the matrix model. The cylinder keeps a history of the completed job-agents, ${}^{a_{j_c}}c_{c_0}$, a list of the job-agents that have started, ${}^{a_{j_s}}c_{c_0}$, the released resources, ${}^{r_{j_s}}c_{c_0}$, and the resources that are currently available, ${}^{r_{j_c}}c_{c_0}$. Based on the system's current state and the environs, the $\mathbf{Q}$ matrices are used to obtain the next set of conditions which in turn will be used by the $\mathbf{S}$ and $\mathbf{U}$ matrices to release the resources required for the starting of a job-agent. Once a job-agent is complete, the $\mathbf{Q}$ matrices are once again used to obtain the next set of conditions. The process is iterative.

The HLJC can learn about the available resources by two means: its own sensors ${}_xc_{V_0}$ and by receiving information from the other agents through ${}_xc_{T_0}$. The information about

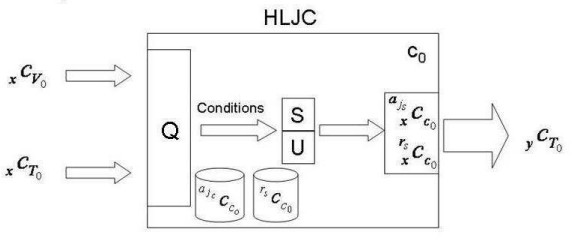

Fig. 3. HLJC with Matrix-based Supervisory Controller

the completion of job-agents usually is received directly from the agents through a ${}_yc_{T_j} \rightarrow {}_xc_{T_0}$ transmission. The information about which job-agent should be currently started is dispatched through ${}_yc_{T_0}$ and received by the agent a_j through ${}_xc_{T_j}$. The matrix equations define the structure of the transition functions, as they are used in the computation of their arguments.

V. Exception Handling

Given the occurrence of some condition that changes the normal flow of execution, a mechanism designed to handle this exception ought to be present. Depending on the situation, the handler may later resume the execution at the original location using the saved information to restore the original state. Errors (which require handling), however, are abundant in nature and cannot be catered to entirely. Therefore, it is apt to encompass these errors in differentiated classes [12]:

- non-fatal (caused by computational problems or wrong arguments of commands)
- fatal (caused by malfunction of the resources)
- system (caused by control system disintegration – the system is rendered useless)

By defining these error classes, the intricacy of exception handling is somewhat reduced.

A. Error Transmission

With an error state present in each job-agent (Fig. 2) and the introduction of the error classes, the domain of this state can now be defined as

$${}^{E}c_{c_j} \in \{E_{NF}, E_F, E_S, E_{NE}\}, \tag{18}$$

where NF = Non-Fatal, F = Fatal, S = System and NE = No Error.

Upon triggering an error condition, the current global state of the job-agent will change to that of the error state which should contain the class of error that is produced. The job-agent's immediate task, given the next time instant, would be to transmit this information to the HLJC. The control system of an agent a_k, in its error state E, uses

$${}_xc_k^i = \langle c_{c_k}^i, {}_xc_{e_k}^i, {}_xc_{V_k}^i, {}_xc_{T_k}^i \rangle, \tag{19}$$

to produce

$${}_yc_k^{i+1} = \langle c_{c_k}^{i+1}, {}_yc_{e_k}^{i+1}, {}_yc_{V_k}^{i+1}, {}_yc_{T_k}^{i+1} \rangle, \tag{20}$$

where the components: ${}_xc_{e_k}$, ${}_xc_{V_k}$ and ${}_xc_{T_k}$ provide information about ${}^{E}c_{c_k}$. The HLJC then instructs the job-agent of its next step.

There are a few standard methods of treating error recovery depending on the error class

- retrying the action that failed,
- retrying the action that failed, but with a different set of parameter values,
- trying another action

It should be noted that the classes of errors pertain to the ability of the system to retain its own composure and not the ability to correct them. For instance, a non-fatal error detected as a computational error (e.g., negative argument of a sqrt function within inverse kinematic procedure), but resulting from an object that is to be grasped being out of the workspace, cannot be corrected by the system (the object is simply too far), but the system should retain such a state that it will be able to execute other actions. The ability of correcting fatal errors depends on the redundancy of the system, e.g., malfunction of the manipulator in a single robot system cannot be remedied, but in a multi-robot one, it can sometimes be corrected. Thus, whether errors can be handled depends also on the overall structure of the system. Nevertheless, after a fatal error, the system should be left in such a state that at least it is able to inform the operator about the reason for its malfunction - it should not disintegrate. System errors cannot be dealt with by the above described methods. They are caused by the disintegration of the control system itself, so it is not realistic to assume that the inter process communication will be intact at that moment.

VI. An Application

As an initial test-bed for the framework, a simple application involving a single mobile manipulator (MM) was carried out. The mobile manipulator consists of a PA-10 mounted on a mobile base (Fig. 4) which was designed and assembled as a collaborative project between the National University of Singapore and the Singapore Institute of Manufacturing Technology.

The MM was engaged to move to a specified location upon which a token will be passed to it. Once the token is received, the MM navigates through some obstacles using the SICK laser sensor. After completing obstacle avoidance, the MM then moves to another specified location allowing the insertion of the token into a slot. Upon successful entry, a user then attaches a spraying mechanism to the MM's gripper after which a star will be sprayed at a designated location.

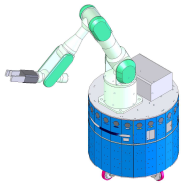

Fig. 4. PA-10 on a mobile base

A. Task Sequence

Goal based decomposition of the task implies a description of the job-agents, which consists of a composition of transition functions. The job-agents are:

1) MM at Location 1, Invert arm, open gripper
2) MM at Location 2, Procure token
3) Token received, Begin obstacle avoidance
4) MM at Location 3, Slot the token
5) MM at Location 4, Procure spraying mechanism
6) MM at Location 5, Draw the star

In the event that a job-agent cannot execute itself, the exception handling module will come forth. Upon unsuccessful handling, a user will be prompted to intervene.

B. Resource Assignment

For the task at hand, three resources are available for procurement:

- mobile manipulator (MM)
- a token
- a spraying mechanism

The MM is assigned to each of the six job-agents signifying the allowable resource it can procure to execute itself. In addition to the MM, job-agents 2 and 4 are also allowed to procure the token and spraying mechanism respectively. The issue of when a resource is procured is decided by the matrix model in the HLJC.

C. Matrix Model

For the sake of brevity, the equation depicting the task (12) will not be discussed (for a theoretical insight, see [8]).A detailed mathematical form (15) of job-agent execution will instead be shown:

$$\begin{bmatrix} \bar{a}_{1_s} \\ \bar{a}_{2_s} \\ \bar{a}_{3_s} \\ \bar{a}_{4_s} \\ \bar{a}_{5_s} \\ \bar{a}_{6_s} \end{bmatrix} = \begin{bmatrix} I_{6\times 6} & 0_{6\times 1} \end{bmatrix} \otimes \begin{bmatrix} \bar{x}_1 \\ \bar{x}_2 \\ \bar{x}_3 \\ \bar{x}_4 \\ \bar{x}_5 \\ \bar{x}_6 \\ \bar{x}_7 \end{bmatrix} \oplus I \otimes \begin{bmatrix} 0 \\ \bar{a}_{1_c} \\ \bar{a}_{2_c} \\ \bar{a}_{3_c} \\ \bar{a}_{4_c} \\ \bar{a}_{5_c} \end{bmatrix}, \tag{21}$$

where

- $\mathbf{a}_{k_s}$, $k = 1, .., 6$ represents each of the job-agents that has yet to execute
- $\mathbf{x}_p$, $p = 1, .., 7$ represents each of the conditions that has to be satisfied in (12). The number of conditions

is always larger than the number of job-agents by one with the last condition signifying task completion.

- $\mathbf{a}_{k_c}$ represents the completed job-agents

Say for example that condition $x_2 = 1$ (satisfied) and job-agent $a_{1_c} = 0$ (incomplete). The components of the conditions are found in (12). From (21), and considering only row $\bar{a}_{2_s}$ followed by De Morgan's law,

$$\begin{aligned} \bar{a}_{2_s} &= \bar{x}_2 \oplus \bar{a}_{1_c} \\ a_{2_s} &= x_2 \otimes a_{1_c} = 0 \end{aligned} \tag{22}$$

Since $a_{2_s} = 0$, job-agent a_2 cannot start. Job-agent a_2 can only start when condition $x_2 = 1$ and job-agent a_1 is complete ($a_{1_c} = 1$). Via iteration, the conditions and job-agent completions will be satisfied stepwise leading to the completion of the task.

VII. CONCLUSIONS AND FUTURE WORKS

A. Conclusions

The general structure of an agent (Fig. 1) does not change with a differing task. However, the class of tasks that can be executed by the agent is limited by the resources available. The resources influence the specific structure of the components of c_j as defined by (1). Those components do not change with the modification of the task within a class (as defined by the resources needed). Here only the transition functions (5) change their form. All this shows us the extent to which software implementation of the agent will have to be modified with the change of the task. New resources require a change in the definition of components of (1). If the resources remain unchanged, the structure of components of (1) remain unaltered and only the definitions of transition functions (5) undergo modification – usually this is the case, because hardware of the system changes infrequently. Moreover, the presented formalism decomposes the system thus facilitating modularization of the software and that promotes its reuse.

Viewing a job as an agent introduces a paradigm shift from the conventional perception of a job. By imparting some level of intelligence in terms of the resources it can procure, jobs, in their abstract form, become fairly autonomous nevertheless still coordinated with the presence of a high level job coordinator (matrix-based supervisory controller).

The overall framework deals with systems consisting of multiple embodied agents, influencing the environment through effectors, gathering information from the environment through sensors and communicating with other agents through communication channels.

Formalization introduces rigor into the discussion of the structure of embodied agent controllers. This structures the implementation of a programming framework, and that in turn makes the coding of specific controllers much easier, from the point of view of specific tasks that has to be executed.

B. Future Works

Given the task at hand, the authors recognize that the framework cannot be fully appreciated due to the limited number of available resources. This task, however, is merely a proof of concept. In order to test the generality of the proposed framework, its application to multi-robot systems with the intent of executing parallel multiple tasks will be pursued.

REFERENCES

[1] 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*, ser. 5, vol. 10, September 2002, pp. 709–716.

[2] M. Boshernitsan and M. Downes, "Visual programming languages: A survey," Computer Science Division (EECS), University of California Berkeley, Tech. Rep., December 2004.

[3] C.-H. Chien, P. Chang, and V.-W. Soo, "Market-oriented multiple resource scheduling in grid computing environments," in *19th Int. Conf. on Advanced Information Networking and Applications*, vol. 1, March 2005, pp. 867–872.

[4] J. Dyson, N. Griffiths, L. C. Keung, S. Jarvis, and G. Nudd, "Trusting agents for grid computing," in *IEEE Int. Conf. on Systems, Man and Cybernetics*, vol. 4, October 2004, pp. 3187–3192.

[5] V. Giordano, F. Lewis, P. Ballal, and B. Turchiano, "Supervisory controller for task assignment and resource dispatching in mobile wireless sensor networks," in *Int. Journal of Advanced Robotic Systems*, ser. Cutting Edge Robotics, V. Kordic, A. Lazinica, and M. Merdan, Eds. Proliteratur Verlag, 2005.

[6] N. W. Koh, M. H. Ang, and S. Y. Lim, "Implementation of a matrix-based discrete event controller for robotic tasks," in *Asian Conf. for Industrial Automation and Robotics*, May 2005, pp. 410–415.

[7] ——, "Robotic tasks employing an improved matrix-based discrete event controller," in *Int. Symp. on Collaborative Research in Applied Science*, October 2005, pp. 195–202.

[8] ——, "A theoretical insight to the improved matrix-based supervisory controller," in *Int. Conf. on Computational Intelligence, Robotics and Autonomous Systems*. Elsevier (EI), ISSN: 02196131, December 2005.

[9] V. Santos Bottazzi and J. Cruz Fonseca, "Off-line robot programming framework," in *Joint Int. Conf. on Autonomic and Autonomous Systems and Int. Conf. on Networking and Services*, October 2005, pp. 71–76.

[10] D. A. Tacconi and F. L. Lewis, "A new matrix model for discrete event systems: Application to simulation," in *IEEE Control Systems*, ser. 5, vol. 17, October 1997, pp. 62–71.

[11] E. Wang and R. Wang, "Using legos and robolab (labview) with elementary school children," in *31st Annual Frontiers in Education Conference*, vol. 1, October 2001, pp. T2E–T11.

[12] C. Zielinski, "Reaction to errors in robot systems," in *Int. Workshop on Robot Motion and Control, RoMoCo*, November 2002, pp. 201–208.

[13] ——, "Formal approach to the design of robot programming frameworks: The behavioural control case," in *Bulletin of the Polish Academy of Sciences*, vol. 53, no. 1, 2005, pp. 1–11.

[14] ——, "Transition-function based approach to structuring robot control software," in *Robot Motion and Control: Recent Developments, Lecture Notes in Control and Information Sciences, Vol.335*, K.Kozlowski, Ed. Springer Verlag, 2006, pp. 265–286.