A Mobile Agent-Based Framework for Flexible Automation Systems

Stephen S. Nestinger, *Member, IEEE*, Bo Chen, *Member, IEEE*, and Harry H. Cheng, *Senior Member, IEEE*

***Abstract*—Modern manufacturing systems are increasingly becoming highly dynamic due to the integration with advanced information technology in response to rapid changes in products and market conditions. A more flexible platform is critically needed for developing a new generation of manufacturing systems in order to address the challenges of uncertainty and flexibility requirements. This paper presents a mobile agent-based framework that supports dynamic deployment of control algorithms and tasks in automation systems. The framework is based on a mobile agent system called Mobile-C. It uses Ch, an embeddable interpretive C/C++ environment for mobile agent execution. Since Ch has been ported to most existing computing platforms, the framework can control automation systems that work in different operating systems. This mobile agent-based framework has been applied to the control of an automation work cell. Using an automaton package in Ch as a middleware, automation tasks can be described as high-level control programs and are portable to heterogeneous mechatronic devices that comprise the automation cell. The validation of the dynamic deployment of different tasks has been conducted in an experimental automation work cell that consists of a Puma 560, an IBM 7575, and a conveyor system. The results show that the mobile agent approach can effectively deploy and execute new control algorithms and tasks as mobile agents on any subsystem in a network.**

***Index Terms*—Automation work cell, mobile agents, robotics, system integration.**

I. Introduction

THE MANUFACTURING sector has been undergoing a hi-tech industrial revolution with information technology as the dominant driving force [1]. In the results of a study completed by the National Research Council [2], the integration of information throughout the enterprise for making effective decisions and the ability to dynamically reconfigure manufacturing enterprises were identified as two of six "grand" challenges for manufacturing. The technology of dynamic agile manufacturing systems was also identified as the number one priority technology to meet the grand challenges. The same views are supported by the study carried out by the Next Generation Manufacturing Project funded by National Science Foundation (NSF) and other federal agencies and headed by a coordinating council drawn from the manufacturing industries. From their report [3], the development and implementation of agile and scalable manufacturing systems are important preliminary steps in achieving the rapid response of production systems. Dynamic agile manufacturing systems are defined as adaptable, integrated equipment, processes, and systems that can be readily reconfigured for a wide range of customer requirements for products, features, and services [2]. To realize agile manufacturing systems, certain key enabling technologies, such as modular machines and open architecture control and information systems must be developed [4], [5].

Traditional centralized and sequential manufacturing schemes for planning, scheduling, and mechanism control have been found to be inflexible and incapable of responding to quick changes in production styles and requirements [6]–[8]. This can be seen in the current trend towards decentralized agent-based approaches to solve complexities introduced by quickly changing market requirements [9] and facilitated by the individualistic desires of customers who choose to participate in the design and production processes [10]. Automation systems are innately heterogeneous and manufactures were intertwined in creating open architecture based flexible and agile manufacturing systems. In previous decades, a large number of researchers have attempted to apply agent technology to manufacturing enterprise integration [11], supply chain management [12], manufacturing design [13], network management [14], planning, scheduling and control [15], [16], material handling, and holonic manufacturing systems [17]. However, very few research was focused on low-level automation system integration and control. An automation system software infrastructure should allow for quick and easy integration of new automatons. In order to increase maintainability, legacy compatibility, flexibility, scalability, and modularity, the low-level automation system infrastructure should be independent from the higher level agent-based systems. To follow suite with the decentralized trend, the low-level infrastructure should be an agent-based system that follows an agent standard to allow for inter-agent system communication.

Most existing agent applications in manufacturing automation systems rely on the use of stationary agents [18]–[23]. Although stationary agents allow for decentralized control, with the increased use of information technology, the saturation of network bandwidth in a distributed automation environment could reduce data security, integrity and timeliness, and render IT support ineffective [24]. Stationary agents also have a limited ability to deal with uncertainty in dynamic environments.

Manuscript received March 19, 2009; revised October 5, 2009; accepted October 19, 2009. Date of publication December 28, 2009; date of current version December 15, 2010. Recommended by Technical Editor Y. Li.

S. S. Nestinger is with the Department of Mechanical Engineering, Worcester Polytechnic Institute, Worcester, MA 01609-2280 USA (e-mail: ssnestinger@wpi.edu).

B. Chen is with the Department of Mechanical Engineering-Engineering Mechanics and the Department of Electrical and Computer Engineering, Michigan Technological University, Houghton, MI 49931 USA (e-mail: bochen@mtu.edu).

H. H. Cheng is with the Department of Mechanical and Aerospace Engineering, University of California, Davis, CA 95616 USA (e-mail: hhcheng@ucdavis.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TMECH.2009.2036169

In contrast, a mobile agent can travel between different execution environments in a network [25]. Mobile agents can be created dynamically during runtime and dispatched to remote systems to perform tasks with the most up-to-date code. Therefore, the mobility of mobile agents provides large-scale distributed applications with significant flexibility and adaptability to deal with unforeseen system perturbations and reduces required network bandwidth. Many mobile agent systems (MASs) have been developed due to an increasing interest in mobile agent technology [26]–[29]. Most of them were developed to support Java mobile agents. However, low-level drivers for automatons are typically written in C. In order to increase open architecture compatibility, it is desirable to use an agent language that is based on the same language as the low-level drivers. Although there were a few agent systems that support C/C++ mobile agents, they are not compliant with either of the two international agent standards: Foundation for Intelligent Physical Agents (FIPA) [30] and MAS Interoperability Facility (MASIF) [31].

With any given real-time manufacturing automation system, automaton program compilation presents a serious problem. In real-time systems, the external environment may be different for each execution. Therefore, the testing scenario may not necessarily be repeatable. During debugging and testing, it is impractical to restart an automation program from the very beginning every time a change is made or a problem is diagnosed. It is well-known that for real-time operation of many mechatronic systems, the language environment has to be interpretive with a quick system response [32], [33]. One sensible solution is to use an open and high-level interpretive automata programming environment that provides the ability to easily integrate heterogeneous hardware.

Middlewares are often used to facilitate control coordination and integration of new hardware in heterogeneous systems [34]. Many middleware have emerged including Carnegie Mellon Robot Navigation (CARMEN) Toolkit [35], Mobile and Autonomous Robot Integration Environment (MARIE) [36], Player/Stage [37], [38], Adaptive Communication Environment (ACE)/TAO [39], and MIRO [40]. Some are based on standard communication models, such as CORBA, while others were implemented without following any standards in order to provide advanced functionality. However, these middleware are not interpretive and do not lend themselves to interactive command or function execution for rapid prototyping [8]. To achieve high-level automation programming and enhance the portability of control programs, we developed a scriptable object-oriented automaton middleware. With an automaton middleware, the heterogeneity of automata is hidden within the implementation of the middleware. A control program written for one automaton can potentially directly control another one with the same control functionality with little or no modification.

This paper presents a mobile agent-based framework for flexible automation systems. The framework is based on a MAS called Mobile-C [41]–[43]. Mobile-C is an IEEE FIPA standards compliant MAS supporting mobile C/C++ agents. Mobile-C can also be used in stationary multiagent system applications. Compliance with IEEE FIPA standards ensures the interoperability of Mobile-C in a heterogeneous network containing other FIPA compliant agent systems such as higher level MASs with focus on planning, processing, inter-enterprise communication, etc. Verification of the flexible automation system using a MAS is accomplished through an experiment with a heterogeneous automation work cell comprising of two manipulators, a Puma 560 and an IBM 7575 robotic manipulator, and a conveyor system. The robotic manipulators have been retrofitted with an open architecture. The automation work cell uses Ch as an interpretive C/C++ automata programming environment. Using scriptable C/C++ as an automaton language allows for a direct interface with low-level device drivers. This makes it easier to integrate new sensors and mechatronic devices into automation systems.

The rest of this paper is organized as follows. Section II introduces an automata programming environment for automation systems. In this section, an interpretive C/C++ automaton programming language and an automaton control middleware are discussed. Examples demonstrating the functionality and power of the automata programming environment are provided. Section III presents a mobile agent-based automation system framework. Section IV provides framework validation through a retrofitted experimental automation work cell. Section V briefly discusses the difference between the mobile agent-based approach and other standard approaches regarding automation systems. Finally, conclusions are drawn in Section VI.

II. Automata Programming Environment

Dynamic agile machines require a highly malleable software architecture to support a rapid change in production. Robot manipulators are common manufacturing tools due to their versatility. All commercial robot manipulators can be programmed to run in their proprietary robot language. For example, a Puma 560 can be programmed using the robot language VAL II, whereas an IBM 7575 can be programmed using AML. Having to deal with multiple-robot programming languages makes it difficult to create cooperative control programs for both manipulators under different kinematic configurations, especially if they are integrated to form an automation work cell. For heterogeneous multi-robot systems, robot programmers have to know a variety of robot programming languages, which may stagnate a project when a new and unfamiliar robot is introduced into the system. An open and high-level robot programming environment should hide the heterogeneity of a robotic system and should follow a standard and widely used programming language. This environment should also be a part of an open architecture integration environment that allows for the integration of mechatronic systems in agile manufacturing, interactive motion control, rapid prototyping, and Web-based remote motion control. Having an interactive environment allows robot programmers to step through motion sequences and acquire immediate feedback from the system. During debugging and testing, it is undesirable to restart a robot program from the very beginning every time a change is made or a problem is diagnosed. A language called Ch is designed to be such an open and high-level automaton programming language [44].

TABLE I
PUBLIC MEMBER FUNCTIONS FOR CONTROL AND KINEMATICS

Robot class public member functions	Purpose
CRobot(int robotnum, char runtype)	Class constructor
~CRobot(int robot_type, char run_type)	Class destructor
int Setup()	Setup the robot
int DacsCheck()	Measure output DACs of the robot
int Calibrate()	Calibrate the robot
int MoveZero()	Move to the zero position
int MoveReady()	Move to the ready position
int MoveHome()	Move to the home position
int MoveWait()	Wait until motion is complete
int MoveLine(double start[4][4], end[4][4], int steps, char conf)	Move the end effector in a line using intermediate steps
int JointSpeed(int speed[])	Set speed for each joint of the robot
int Drive(double jangle[])	Move joints specified in degrees
int InverseKinematics(double T[4][4], jold[], jnew[], int conf)	Calculate the robot inverse kinematics
int GripperOpen()	Open the robot gripper
int GripperClose()	Close the robot gripper
int Disable()	Disable the robot

A. Interpretive C/C++ Automaton Programming Language

Ch, originally developed by Cheng [45], [46], is an embeddable C/C++ interpreter [47] developed as an integrated programming environment for cross platform scripting [45], and numerical and embedded computing. Ch is an extension and enhancement of the widely used C computing environment. Being platform independent, a Ch program developed on one platform can be readily executed on different platforms without recompiling and linking. The ease in use of Ch for the interactive control and integration of heterogeneous mechatronic and robotic systems has already been verified in [48], [49]. Functions, commands, and scripts in Ch are the basic building blocks for high-level robot programming. A *function file* in Ch is a C program that contains at least one function definition. In the Ch programming environment, functions defined using function files are treated as if they were built-in system functions. With Ch function files, automaton programs are created not by writing large programs from scratch, but are instead combined from relatively small components. These components are small and concentrate on simple commands so that they are easy to build, understand, describe and maintain. The command-line argument interface in Ch is C compatible. Statements, functions, and commands can be grouped as a *script file* or a *script* in Ch. A script can also contain scripts written in other shells such as the C shell, Bourne shell, and Korn shell.

As an embeddable C/C++ interpreter, Ch can be embedded into a C/C++ application to enhance prototyping capabilities and increase flexibility. With embedded Ch, users can load scripts into their application incrementally and execute them dynamically. As a built-in C scripting engine, embedding Ch can relieve automation programmers from the burden of developing and maintaining a scripting language or interpreter and instead focus on the core research or application of the system. Embedded Ch has a small footprint to meet the limited memory capacities of various embedded and mechatronic systems for automation.

Ch contains all of the same functionality provided in MATLAB and has been shown to be 2.5 times faster in execution [50]. Ch has also been integrated with RecurDyn/CoLink, a signal-flow simulation tool with a graphical block diagram interface for control program construction [51].

B. Automaton Middleware

An automata typically consists of mechatronic devices encompassing a combination of stepper motors, dc and ac motors, laser and vision sensors, force and torque sensors, tactile sensors, pneumatic grippers, etc. Integrating these devices into a mechatronic system can be cumbersome. Programmers must be familiar with address values for each sensor and the low-level function calls required to access those addresses. Automaton programming languages should remove the necessity of having to deal with low-level programming and device access. This becomes especially important when trying to integrate mechatronic devices from different vendors and maintain consistent interfaces external to machinery. Since most low-level drivers are written in C, it is desirable that the middleware environment that interfaces with these drivers be C compatible. This feature is achieved in the Ch automata programming environment through an automaton middleware. The middleware provides higher programming functionality to users while keeping the interface requirements with low-level drivers hidden. The middleware also keeps system heterogeneity hidden from the user so that automation programs and functions are portable and readily usable in applications with different mechatronic systems. This is accomplished by hiding functions tailored to specific robotic systems under an API with functionality common to all robots in an automation system. When using multiple robotic systems, users are relieved from having to learn multiple-robot programming languages.

The developed automaton middleware, available at [52], provides commonly used functions for robot manipulator programming and currently supports two robotic manipulators, the Puma 560 and IBM 7575 with different kinematic configurations, and a conveyor system. The middleware is object-oriented and accessible through the use of a robot class. The public member functions used for control and kinematics are listed in Table I. Robot-dependent information such as Denavit–Hartenberg parameters, calibration data, and servo control parameters are stored in configuration header files. New mechatronic systems, functionality, and kinematic solutions [53], [54] can be easily added by either incorporating new member functions or modifying exiting ones. For example, if a new robotic system is added to the system, a new inverse kinematics member function for that system can

be easily integrated without the need of linking or compiling. Any necessary modification is accomplished by simply opening the required file in an editor, modifying the contents of the file, and saving the results. The capabilities of the available member functions in the robot middleware are highlighted shortly.

The CRobot() member function is the constructor for the CRobot class. The first argument it takes, `int robot_type`, defines which robot object will be instantiated such as either a Puma 560, IBM 7575 or conveyor object. The second argument, `char run_type`, determines whether the system will operate in simulation or real-time mode. The member function ~CRobot() is the destructor for the CRobot class. When a program exits, it will call the class destructor to deal with any required clean up.

The member function Setup() sends low-level robot-dependent information to the servo control board for each motor of the robot. When this function is executed, servo control is enabled. The member function DacsCheck() checks the servo command signal of each channel by comparing them with the maximum permissible digital-to-analog converter (DAC) voltage. The member function Calibrate() contains calibration protocols and movements for finding the absolute position. The member functions GripperOpen() and GripperClose() control the status of the gripper attached to the robot.

The member functions MoveZero(), MoveReady(), and MoveHome() move the robot to predefined positions. The member function MoveLine() moves the end-effector in a line with given number of intermediate steps. The member function Drive() moves each joint of the robot by the specified angle. By repeatedly calling this function, the robot can move in a desired trajectory. The member function InverseKinematics() performs robot inverse kinematics and passes back the corresponding joint angles for a position specified in a 4×4 homogeneous transformation matrix. The joint angles are then used to drive the robot to the desired position. When a drive command is issued to the servo board, the motion control is executed and the joints put into motion. The member function MoveWait() causes the motion control program to dwell at the calling position until the manipulator is at the desired location.

C. C/C++ Automaton Programming Examples

The following sections provide some examples on how to utilize the described Ch automaton middleware. The first example shows the single command programmability of the middleware while the second example shows how member functions can be sequenced and placed into a script.

1) Interactive Command Line Execution: Ch provides robot programmers the ability to try out robot control commands without having to place them into a strict sequence. This removes the cumbersome requirement of having to restart a robot program when something goes wrong. The use of the Ch robot middleware allows users to create an object and call middleware functions directly from the command line. An example of interactive command line robot control is shown in Fig. 1. The first command,

```
chparse ./include/robot.h
```

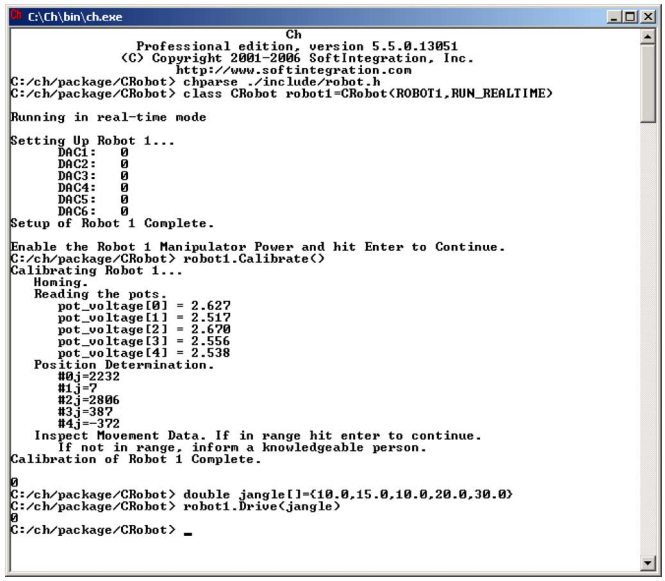

Fig. 1. Interactive command line automaton control session.

```
#include <robot.h>

class CRobot robot1=CRobot(ROBOT1,RUN_REALTIME);
robot1.Calibrate();
double jangle[]={10.0,15.0,10.0,20.0,30.0};
robot1.Drive(jangle);
```

Program. 1. A script containing automaton control commands.

parses the middleware header file that contains the robot class into Ch memory. The second command,

```
class CRobot robot1 = CRobot(ROBOT1, RUN_REALTIME)
```

creates an object `robot1` of type `class CRobot` and passes information to the class constructor used to initialize private data members. The first argument, `ROBOT1`, dictates that this object will be of type Puma 560. The second argument, `RUN_REALTIME`, dictates that the system will be run in real-time instead of simulation. The next command,

```
robot1.Calibrate()
```

calibrates the robot by homing it. The command,

```
double jangle[] = {10.0, 15.0, 10.0, 20.0, 30.0}
```

declares an array of type double called jangle to store the joint angles that the Puma 560 will be driven to. The final command,

```
robot1.Drive(jangle)
```

drives the robot to the desired joint angles. Users can continuously try out different commands until they have a desired sequence.

2) Automata Program and Script Execution: Once a programmer has gone through each command and produced a desired sequence of commands, the commands can be placed into a script that can be run in Ch. A script containing the commands used in the previous section is shown in Program 1.

From these examples, we can clearly see that robot tasks or operations can be described in a very high-level with the Ch

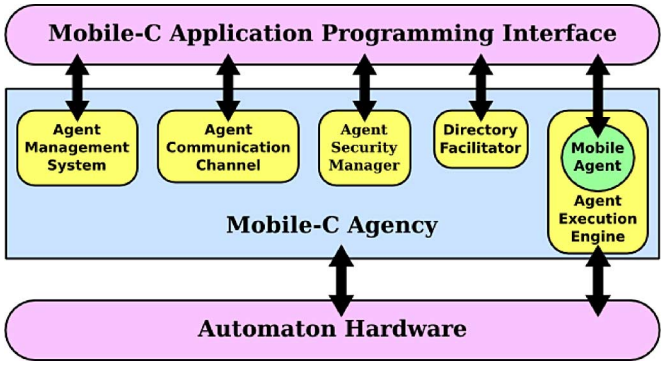

Fig. 2. Architecture of the Mobile-C library.

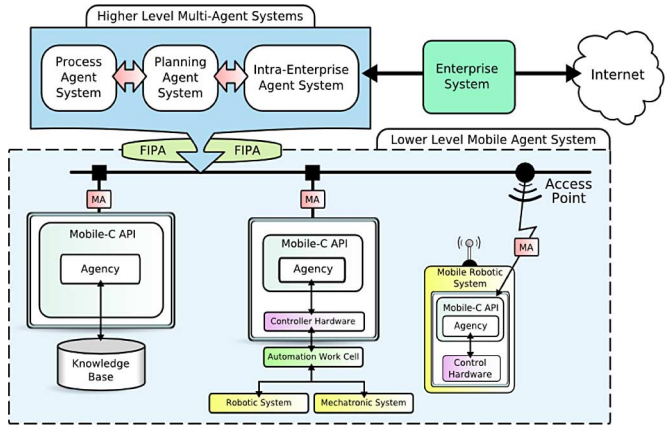

Fig. 3. Mobile agent-based multi-robot systems architecture.

automaton middleware. This is desirable for the code mobility in a MAS.

III. Mobile Agent-Based Automation Systems

In automation work cells, mobile agents allow for the dynamic deployment of new tasks and reconfiguration of control algorithms and parameters. A flexible automation system requires an open architecture, integration environment that is mechatronic device and computer-platform independent. In order to interact with higher level agent systems involved with processing, planning, inter-enterprise, etc., the agent system should comply with inter-agent and intra-agent system communication standards.

A. Mobile-C Library

Mobile-C [41], [42] was originally developed as a standalone, FIPA compliant MAS with a primary intention of fitting into applications with networked mechatronic and embedded systems involving low-level hardware. Since most of these systems are written in C/C++, Mobile-C chooses C/C++ as the mobile agent language for easy interfacing with low-level control programs and the underlying hardware. In addition, Mobile-C uses embedded Ch to support the execution of C/C++ mobile agents. For ease of integrating Mobile-C with applications, a Mobile-C API [43] was developed that allows Mobile-C to be embedded into applications to support C/C++ mobile agents. In addition, the Mobile-C API facilitates the development of a multiagent system that can easily interface with a variety of hardware devices. The architecture of the Mobile-C library is shown in Fig. 2. The components of the Mobile-C library include the Agent Management System (AMS), Agent Communication Channel (ACC), Agent Security Manager (ASM), the Directory Facilitator (DF), and the Agent Execution Engine (AEE).

The AMS manages the life cycle of the agents. It controls creation, registration, retirement, migration, and persistence of agents. The ACC routes messages between local and remote entities, realizing messages using an agent communication language (ACL). All interactions can be performed via ACL message exchange. The ASM is responsible for maintaining security policies for the platform and infrastructure, such as communications and transport-level security. The DF serves yellow page services. Agents in the system can register their services with DF to provide to the community. They can also look up required services with the DF. The AEE serves as the workhorse for the mobile agents. Mobile agents must reside inside an engine to execute. Since the AEE of Mobile-C is based on Ch, C code can readily be used as Mobile-C agent code. The mobile agent code is wrapped into a mobile agent message with a FIPA ACL in extensible markup language (XML). The XML mobile agent message contains other encapsulated agent data along with the agent C code. The encapsulated agent data include agent states, an itinerary, and tasks.

B. Architecture for Mobile Agent-Based Automata

Fig. 3 depicts the mobile agent-based architecture for a multi-robot system comprising of a vast set of robots and spans multiple networks. The architecture demonstrates the possibility of having geographically distributed robotic systems connected via a WAN and is divided into two levels: high-level systems and a low-level MAS. The high-level systems comprise of enterprise information systems and high-level multiagent systems. The high-level multiagent systems may involve inter-enterprise management, processing, planning, etc., and disseminate tasks to the lower level MAS. Communication between the high-level agent systems and the low level MAS is facilitated with FIPA ACL compliance. The low-level MAS encompasses all robotic-hardware- and hardware-related databases including mobile and static robotic and mechatronic systems. Each system within the lower level is assumed to contain a mobile agency capable of receiving and sending mobile agents. Mobile devices attach to system infrastructure through access points. The dynamic knowledge bases may act as common context where mobile agents can gain access to global information or modify the contents to update the world state. The knowledge bases may also contain robot-specific configuration data or other mechanism required to provide autonomic functionality. The knowledge bases can be geographically distributed along different nodes or across multiple networks. Due to the ease in new system integration with the described MAS, if a production unit goes down, a new one can be quickly integrated into the system without a lot of time overhead. A mobile agent can be sent over to the new unit with the required knowledge and production can quickly

```
<?xml version="1.0"?>
<!DOCTYPE myMessage SYSTEM "mobilec.dtd">
<MOBILEC_MESSAGE>
<MESSAGE message="MOBILE_AGENT">
<MOBILE_AGENT>
<AGENT_DATA>
<NAME>mobagent1</NAME>
<OWNER>IEL</OWNER>
<HOME>localhost:5050</HOME>
<TASKS task="1" num="0">
<TASK num="0" complete="0" server="localhost:5051">
<DATA dim="0" name="no-return" >
</DATA>
</TASK>
<AGENT_CODE>
<![CDATA[
  // Mobile Agent C Code
  #include <stdio.h>
  #include "robot.h"

  int main(void)
  {
    // Instanciate CRobot class
    class CRobot
            ibm = CRobot(ROBOT2,    RUN_REALTIME),
           puma = CRobot(ROBOT1,    RUN_REALTIME),
       conveyor = CRobot(CONVEYOR1, RUN_REALTIME);

    // Calibrate IBM robot and conveyor
    status = puma.Calibrate();
    status = ibm.Calibrate();
    status = conveyor.Calibrate();

    // Move IBM robot to the ready position
    status = ibm.MoveReady();
    status = ibm.MoveWait();

    // Do assembly cell operations
    while(1) {...}

    // Disable all
    status = puma.Disable();
    status = ibm.Disable();
    status = conveyor.Disable();
    return 0;
  }
]]>
</AGENT_CODE>
</TASKS>
</AGENT_DATA>
</MOBILE_AGENT>
</MESSAGE>
</MOBILEC_MESSAGE>
```

Program 2. One of the Mobile-C agents used in the first case study.

resume. If an operation fails or a new production unit or supply is required, a mobile agent can be deployed to pass information up to the high-level agent systems. The information can then passed to the enterprise system for integration with other information systems such as supply chains, front office, production tracking, etc.

C. Mobile Agent Format

The Mobile-C mobile agent is composed of mobile C/Ch agent code encapsulated using XML as shown in Program 2. The agent XML tags consist of the agent's type, name, owner, home, tasks, and C/Ch compatible mobile code.

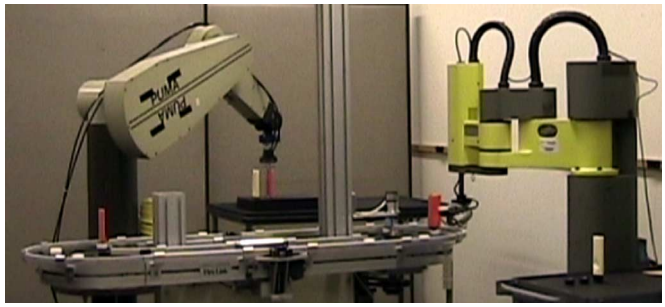

Fig. 4. Experimental automation work cell with a Puma 560 and IBM 7575, and a conveyor system.

IV. Experimental Automation Work Cell

An experimental automation work cell with multiple robots was used to verify the principles behind the mobile agent-based control of automation systems. The automation work cell comprises of Unimation's Puma 560 and IBM's 7575 robotic manipulators and a conveyor system [55], [56]. The robotic systems were retrofitted to comply with open architecture requirements. Initially, the retrofitted robot controller consisted of a Delta Tau Data System's Programmable Multiaxis Controller (PMAC) board seated in a VMEbus computer, a Motorola MVME167, and coordinated by Ch running under the LynxOS real-time operating system from Lynx Real-Time Systems. The robotic cell has since been updated. The retrofitted robot controller now utilizes two Delta Tau Data System's Turbo PMAC2 PCI controllers [57]. The controllers are housed in a PC running Windows XP and coordinated through Ch. Details concerning the update of the retrofitted industrial manipulators are given in the following sections. The mobile agent architecture used with the automation work cell is also described along with an example demonstrating the dynamic deployment of robot tasks via mobile agents to the work cell.

A. Hardware Setup

The hardware configuration of the automation work cell is shown in Fig. 4. The Puma 560 is a widely used general-purpose 6-DOF manipulator. The IBM 7575 is a scara-type robot manipulator with 4-DOF, which are popular in automated chip manufacturing. The automation work cell is setup to execute assembly operations.

B. Control Architecture

The hardware architecture of the automation work cell is shown in Fig. 5. The automation work cell is distributively controlled using two Delta Tau Data System's Turbo PMAC2 PCI controllers. One controller handles the Puma 560 robot manipulator while the other controller handles the IBM 7575 robot manipulator and conveyor system. The PMAC2 cards are housed in a standard PCI capable computer. High-level motion control commands are executed in Ch. Ch interfaces the hardware through Delta Tau Data System's Pcomm32 low-level device drivers.

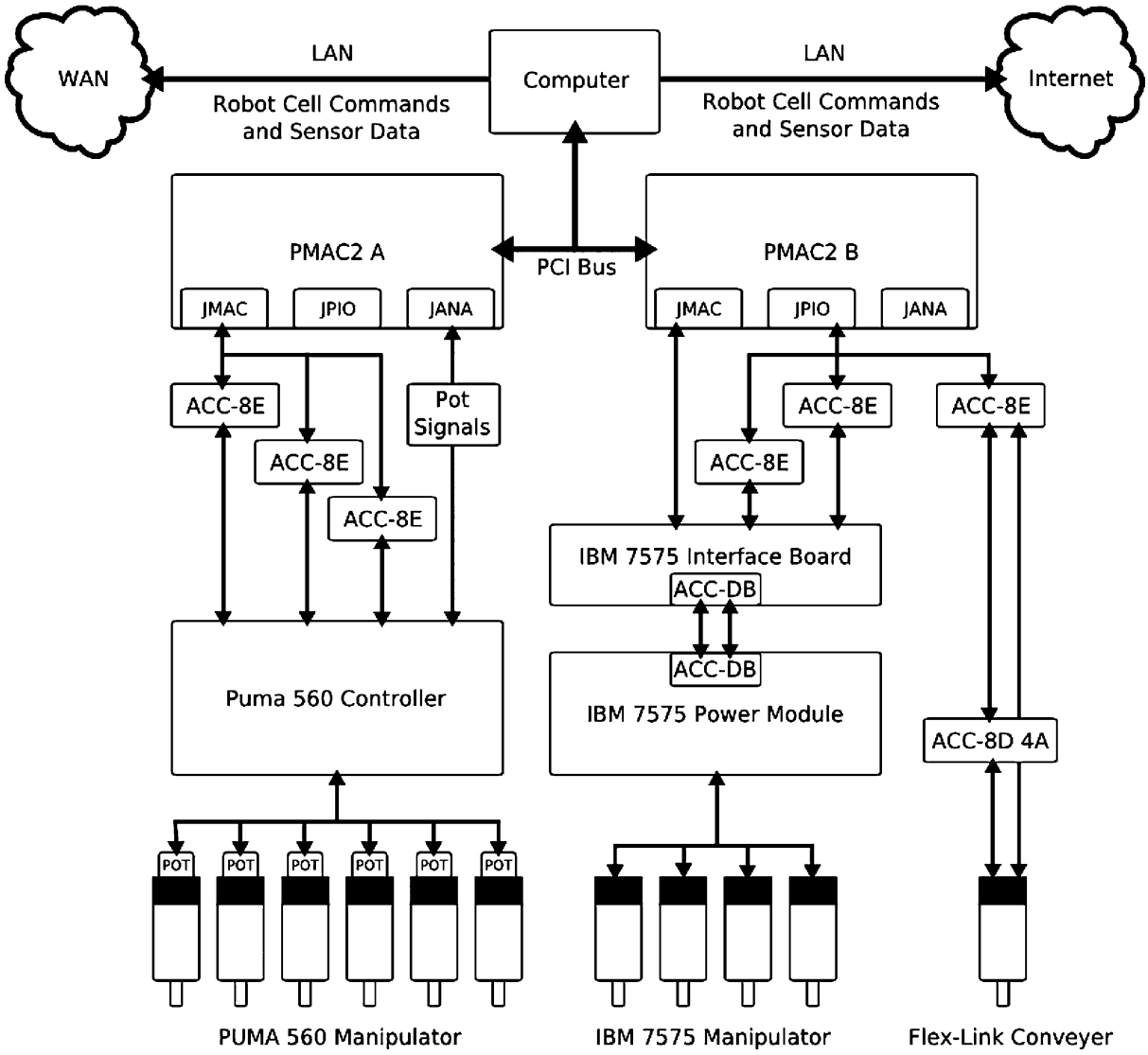

Fig. 5. Hardware architecture of the automation work cell with Puma 560 and IBM 7575 robots.

C. Automation Framework Verification Setup

The experimental automation test bed for validating the mobile agent approach is shown in Fig. 6. The computer system of the automation work cell runs a Mobile-C agency that is ready to receive mobile agents. A user stationed at a remote computer connected on the same network sends mobile agents to the automation work cell system.

A flowchart depicting the deployment and execution of the control mobile agents to the automation work cell is shown in Fig. 7. Initially, the first mobile agent, MA-1, is sent over to the automation work cell. The agent contains the desired control functions that make up the first automation task, Task-1. Once the agent have been received, it begins its execution on the automation work cell computer. The mobile code is written in the Ch language and uses functions from the previously discussed Ch robot middleware to coordinate the motion of the assembly operations of the automation work cell. The automation mobile agent code and its video are available on the Web at [58].

The first mobile agent code simulates operations of an automation assembly and is provided in Program 2. When the automation work cell is started, the two robots and the conveyor system are moved to their ready position after initialization and are calibrated. Then, the IBM 7575 robot is used to pickup parts from a designated location and place them on the conveyor system. Once a part has been placed, the conveyor system will rotate. As the conveyor is rotating into its preset position, the IBM 7575 will then move back to the designated pickup location while the Puma 560 moves to pickup a part from the conveyor. After the Puma 560 has picked up a part, it moves to a drop off location and positions the piece. This is accomplished by calculating the initial and final positions of motion for the two robots. The assembly cycle simulates the assembly procedure of a part. Once a part has gone though its initial stages of manufacturing, it is placed on a conveyor system and brought to either the next stage in fabrication or packaging. The earlier operations are repeated continuously using a while-loop. The motion of the

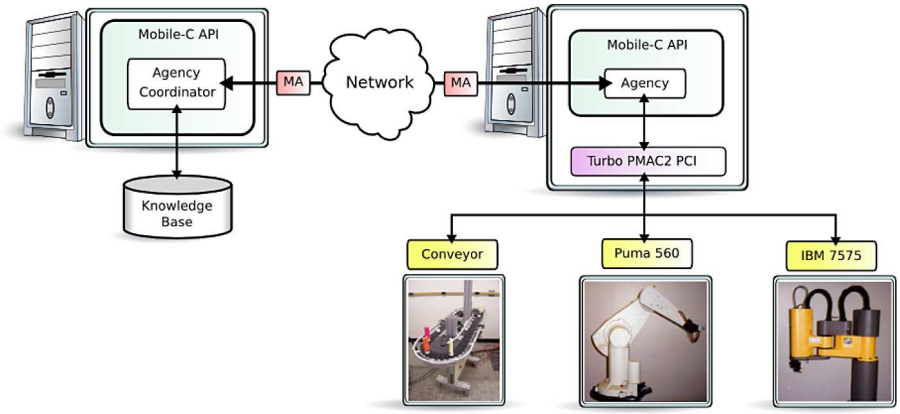

Fig. 6. MAS validation setup.

robots are synchronized to ensure that a robot is fully stopped before proceeding with the next control command.

In order to demonstrate the real-time deployment capabilities of using a MAS, a second mobile agent, MA-2, is sent while the first mobile agent is still executing on the automation work cell. The second mobile agent is sent over to terminate the first mobile agent and perform a new automation task, Task-2. For simplicity and illustrative purposes, Task-2 is the same as Task-1. In order to ensure a safe migration from Task-1 to Task-2, the task termination method is accomplished through the use of a status check at the end of each assembly loop in Task-1. When the Mobile-C agency of the robotic cell receives the second mobile agent, it changes the status of the first agent to neutral. At the end of each assembly loop, the first mobile agent checks its status. If its status has been changed to neutral, it will quit the loop and stop Task-1 in a safe and controlled manner. Limitations on robotic resource use is enforced using a semaphore. The first mobile agent will imitate a semaphore for the automation cell and take control of the cell. After the second mobile agent changes the status of the first mobile agent, it then waits on the semaphore until the resource is free. The first mobile agent releases or frees the resource when it completes.

The validation of the framework using the retrofitted automation work cell demonstrates two important aspects of the framework. Programming two robots with completely different kinematic configurations is quite simple using the robot middleware under Ch. More importantly, even if different computer platforms such as PC-based computer and different mechatronic components such as servo control boards from different vendors were used for our automation work cell, the assembly operations of the automation work cell could be achieved using the same Ch application program. Second, use of a mobile agent-based framework provides the rapid response to changes in automation requirements needed for agile manufacturing. Manufactures are able to terminate currently running automation programs and quickly initiate new ones.

V. Discussions

The following section discuss differences between mobile agent and other standard approaches to automation systems.

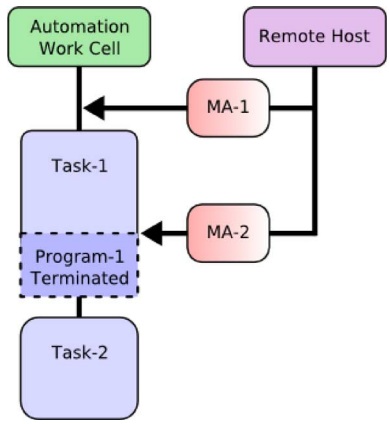

Fig. 7. Flowchart for the MAS deployment.

Many automation system are implemented using a monolithic approach or based on graphical block programming methods. These systems remain inflexible to automated modifications of the automation procedures. In order to modify the automation procedures, the automation system must be stopped while the graphical block program is edited and tested. The developed mobile agent-based framework allows for online reconfiguration of the automation procedures as demonstrated in the automation experimented discussed in Section IV. Mobile agents can be dynamically deployed during runtime allowing for the rapid procedural reconfiguration of an automation system. Mobile agents can also monitor the state of the robots in the automation system and directly and rapidly respond to undesirable perturbations.

VI. Conclusion

This paper presented a flexible automation framework based on mobile agent technology. The framework utilizes Ch as a high-level automata programming environment. Using the Ch automaton control middleware provides automation programmers with the ability to write high-level control programs. The middleware is an object-oriented automaton software package. It hides the heterogeneity of the various possible devices that comprise an automation work cell from application users.

Automaton-dependent information such as Denavit–Hartenberg and inertia parameters of a robot manipulator are stored in configuration files and system implementation modules. Examples that demonstrate the enhancement of the rapid prototyping capabilities of automation programmers by using this C/C++ interpretive environment were given in the article.

A networked architecture is introduced, which allows the use of mobile agents in the low-level automation systems of a manufacturing enterprise. The Mobile-C library is used as the base for this mobile agent-based automata architecture. Use of mobile agents in an automation system allows for the encapsulation of automaton tasks as mobile agents. Dispatching mobile agents to target automatons or automation work cells in the network is done dynamically, which removes the need to halt the system for automata task modification. Mobile-C utilizes Ch as its execution engine, which provides all of the salient features of using Ch as an automata programming environment in a distributed heterogeneous system. The validity of the architecture is proven in an experimental setup with an automation work cell comprised of multiple retrofitted mechatronic systems with a Puma 560, an IBM 7575, and a conveyor system. Future prospects include extending the available functionality of the middleware to provide more sophisticated movements including blended moves and introducing fault-tolerant mechanisms.

REFERENCES

[1] E. Oztemel, C. Kubat, and H. Taskin, "Special issue: Intelligent manufacturing systems: Vision for the future," *J. Intell. Manuf.*, vol. 15, no. 4, p. 415, 2004.

[2] Visionary Manufacturing Challenges for 2020. (1998). Committee on Visionary Manufacturing Challenges, Commission on Engineering and Technical Systems. Nat. Res. Council, Washington, DC [Online]. Available: http://www.nap.edu/readingroom/books/visionary/

[3] *Next Generation Manufacturing: A Framework DOE Action, Agility Forum, Leaders for Manufacturing, and Technologies Enabling Agile Manufacturing,* Next Generation Manuf. Project, Bethlehem, PA, 1997.

[4] M. G. Mehrabi, A. G. Ulsoy, and Y. Koren, "Trends and perspectives in flexible and reconfigurable manufacturing systems," *J. Intell. Manuf.*, vol. 13, no. 2, pp. 135–146, 2002.

[5] P. K. Wright and I. Greenfeld, "Open architecture manufacturing: the impact of open-system computers on self-sustaining machinery and the machine tool industry," in *Proc. Manuf. Int. 1990*, vol. 2, pp. 41–47.

[6] A. Luder, A. Klostermeyer, J. Peschke, A. Bratoukhine, and T. Sauter, "Distributed Automation: PABADIS versus HMS," *IEEE Trans. Ind. Inf.*, vol. 1, no. 1, pp. 31–38, Feb. 2005.

[7] H. Wada and S. Okada, "An autonomous agent approach for manufacturing execution control systems," *Integr. Comput.-Aided Eng.*, vol. 9, no. 3, pp. 251–262, 2002.

[8] H. H. Cheng, F. Proctor, J. L. Michaloski, and W. P. Shackleford, "Real-time computing in open systems for manufacturing," *J. Comput. Inf. Sci. Eng. ASME Trans.*, vol. 1, no. 1, pp. 92–99, Mar. 2001.

[9] N. R. Jennings, "An agent-based approach for building complex software systems—why agent-oriented approaches are well suited for developing complex, distributed systems," *Commun. ACM*, vol. 44, no. 4, pp. 35–41, 2001.

[10] L. M. Camarinha-Mathos and H. Afsarmanesh, "Infrastructures for virtual enterprises: A summary of achievements," in *Proc. IFIP Int. Conf. Infrastructures Virtual Enterprises (PRO-VE)*, 1999, pp. 483–490.

[11] T. Kaihara, "A multiagent-based complex systems approach for dynamic negotiation mechanism in virtual enterprise," *Robot. Comput.-Integr. Manuf.*, vol. 24, pp. 656–663, 2008.

[12] J.-M. Frayret, S. D'amours, A. Rousseau, S. Harvey, and J. Gaudreault, "Agent-based supply-chain planning in the forest products industry," *Int. J. Flexible Manuf. Syst.*, vol. 19, pp. 358–391, 2007.

[13] W. Shen and H. Ghenniwa, "A distributed multidisciplinary design optimization framework based on Web and agents," in *Proc. ASME 2002 DETC Comput. Inf. Eng. Conf.*, Montreal, QC, Canada, pp. 411–418.

[14] C.-Y. Huang, K. Cheng, and A. Holt, "An integrated manufacturing network management framework by using mobile agent," *Int. J. Adv. Manuf. Technol.*, vol. 32, no. 7/8, pp. 822–833, Apr. 2007.

[15] C. V. Trappey, A. J. Trappey, C.-J. Huang, and C. Ku, "The design of a JADE-based autonomous workflow management system for collaborative SoC design," *Expert Syst. Appl.*, vol. 36, pp. 2659–2669, 2009.

[16] F. T. S. Chan and J. Zhang, "A multi-agent-based agile shop floor control system," *Int. J. Adv. Manuf. Technol.*, vol. 19, no. 10, pp. 764–774, 2002.

[17] Holonic manufacturing systems consortium web site. (2002). [Online]. Available: http://hms.ifw.uni-hannover.de

[18] F. T. S. Chan, R. Swarnkar, and M. K. Tiwari, "Infrstructure for coordination of multi-agents in a network-based manufacturing system," *Int. J. Adv. Manuf. Technol.*, vol. 31, pp. 1028–1033, 2007.

[19] F. T. S. Chan, J. Zhang, and P. Li, "Agent- and CORBA-based application integration platform for an agile manufacturing environment," *Int. J. Adv. Manuf. Technol.*, vol. 21, pp. 460–468, 2003.

[20] H. Li, F. Karray, O. Basir, and I. Song, "A framework for coordinated control of multiagent systems and its applications," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 38, no. 3, pp. 534–548, May 2008.

[21] P. Leitao and F. Restivo, "Implementation of a holonic control system in a flexible manufacturing system," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 38, no. 5, pp. 699–709, Sep. 2008.

[22] D. Zhang and A. A. M. K. Lim, "Dynamically integrated manufacturing systems (DIMS): A multiagent approach," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 37, no. 5, pp. 824–850, Sep. 2007.

[23] C.-Y. Yu and H.-P. Huang, "Development of the order fulfillment process in the foundry fab by applying distributed multi-agents on a generic message-passing platform," *IEEE/ASME Trans. Mechatronics*, vol. 6, no. 4, pp. 387–398, Dec. 2001.

[24] T. Papaioannou and J. Edwards, "Building agile system with mobile code," *Auton. Agents Multi-Agent Syst.*, vol. 4, pp. 293–310, 2001.

[25] A. Fuggetta, G. P. Picco, and G. Vigna, "Understanding code mobility," *IEEE Trans. Softw. Eng.*, vol. 24, no. 5, pp. 342–361, May 1998.

[26] J. Baumann, F. Hohl, K. Rothermel, M. Strasser, and W. Theilmann, "MOLE: A mobile agent system," *Softw.-Pract. Exp.*, vol. 32, no. 6, pp. 575–603, 2002.

[27] D. B. Lange and M. Oshima, *Programming and Deploying Java Mobile Agents With Aglets*. Reading, MA: Addison-Wesley, 1998.

[28] H. Peine, "Application and programming experience with the ara mobile agent system," *Softw.-Pract. Exp.*, vol. 32, no. 6, pp. 515–541, 2002.

[29] JADE—Java agent development framework. (1998). [Online]. Available: http://jade.tilab.com/

[30] FIPA: The foundation for intelligent physical agents. (1996). [Online]. Available: http://www.fipa.org/

[31] Object Management Group. Mobile agent system interoperability facilities specification. (1989). [Online]. Available: http://www.omg.org/docs/orbos/97-10-05.pdf

[32] H. H. Cheng, "Real-time manipulation of a hybrid serial-and-parallel-driven redundant industrial manipulator," *Trans. ASME, J. Dyn. Syst., Meas., Control*, vol. 116, pp. 687–701, Dec. 1994.

[33] H. H. Cheng and R. Penkar, "Stacking irregular-sized packages by a robot manipulator," *IEEE Robot. Autom. Mag.*, vol. 2, no. 4, pp. 12–20, Dec. 1995.

[34] N. Mohamed and J. Al-Jaroodi, "Characteristics of middleware for networked collaborative robots," in *Proc. Int. Symp. Collaborative Technol. Syst. (CTS 2008)*, May, pp. 524–531.

[35] M. Montemerlo, N. Roy, and S. Thrun, "Perspectives on standardization in mobile robot programming: The carnegie mellon navigation (CARMEN) toolkit," in *Proc. IEEE/RSJ Int. Conf., Intell. Robots Syst.*, Las Vegas, NV, Oct. 2003, pp. 2436–2441.

[36] C. Cote, D. Letourneau, F. Michaud, J.-M. Valin, Y. Brosseau, C. Raievsky, M. Lemay, and V. Tran, "Code reusability tools for programming mobile robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS 2004)*, Sep. 28–Oct. 2, vol. 2, pp. 1820–1825.

[37] T. H. Collett, B. A. MacDonald, and B. P. Gerkey, "Player 2.0: Toward a practical robot programming framework," in *Proc. Australas. Conf. Robot. Autom. (ACRA 2005)*, Sydney, Australia, Dec. 2005.

[38] B. P. Gerkey, R. T. Vaughan, K. Stoy, A. Howard, G. S. Sukhatme, and M. J. Mataric, "Most valuable player: A robot device server for distributed control," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS 2001)*, Wailea, HI, Oct. 29–Nov. 3, pp. 1226–1231.

[39] D. C. Schmidt, A. Gokhale, T. Harrison, and G. Parulkar, "A high-performance endsystem architecture for real-time CORBA," *IEEE Commun. Mag.*, vol. 14, no. 2, pp. 72–77, Feb. 1997.
[40] H. Utz, S. Sablatnog, S. Enderle, and G. Kraetzschmar, "Miro—Middleware for mobile robot applications," *IEEE Trans. Robot. Autom.*, vol. 18, no. 4, pp. 493–497, Aug. 2002.
[41] B. Chen, H. H. Cheng, and J. Palen, "Mobile-C: A mobile agent platform for mobile C/C++ agents," *Softw.-Pract. Exp.*, vol. 36, no. 15, pp. 1711–1733, Dec. 2006.
[42] Mobile-C: A Multi-Agent Platform for Mobile C/C++ Code. (2005). [Online]. Available: http://www.mobilec.org
[43] Y.-C. Chou, D. Ko, and H. H. Cheng, "An embeddable mobile agent platform supporting runtime code mobility, interaction and coordination of mobile agents and host systems," *Inf. Softw. Technol.*, vol. 52, no. 2, pp. 185–196, Feb. 2009.
[44] H. H. Cheng, "Toward task-level robot programming," in *Proc. 2nd Chin. World Congr. Intell. Control Intell. Autom.*, Xian, China, vol. 1, Jun. 1997, pp. 648–653.
[45] H. H. Cheng, "Ch: A C/C++ interpreter for script computing," *C/C++ User's J.*, vol. 24, no. 1, pp. 6–12, Jan. 2006.
[46] H. H. Cheng, "Handling of complex numbers in the Ch programming language," *Sci. Program.*, vol. 2, no. 3, pp. 76–106, Fall 1993.
[47] Ch—An embeddable C/C++ interpreter. (2001). [Online]. Available: http://www.softintegration.com
[48] S. Nestinger and H. H. Cheng, "Interactive motion control using Ch—An embeddable C/C++ interpreter," *Assem. Autom.*, vol. 24, no. 2, pp. 152–158, 2004.
[49] K. G. Cheetancheri and H. H. Cheng, "Mixed language programming in C/C++ and Java for applications in mechatronic systems," presented at the IEEE/ASME Int. Conf. Mechatronic Embedded Syst. Appl., Beijing, China, Aug. 2006.
[50] Y.-C. Chou, S. S. Nestinger, and H. H. Cheng, "Ch MPI: interpretive parallel computing in C," *IEEE Comput. Sci. Eng.*, to be published.
[51] D. J. Yun, J. H. Choi, S. G. Lee, and H. H. Cheng, "A case study of mechatronic system simulation: Forklift electric control," presented at the 7th Int. Conf. Multibody Syst., San Diego, CA, Aug. 30–Sep. 2 2009, Paper DETC2009-87568.
[52] Ch Robot Package. (2007). [Online]. Available: http://iel.ucdavis.edu/projects/chrobot
[53] L. Jiang, D. Sun, and H. Liu, "An inverse-kinematics table-based solution of a humanoid robot finger with nonlinearly coupled joints," *IEEE/ASME Trans. Mechatronics*, vol. 14, no. 3, pp. 273–281, Jun. 2009.
[54] J. Liu, Y. Zhang, and Z. Li, "Improving the positioning accuracy of a neurosurgical robot system," *IEEE/ASME Trans. Mechatronics*, vol. 12, no. 5, pp. 527–533, Oct. 2007.
[55] X. Hu, J. Pannu, and H. H. Cheng, "Retrofitting an automatic manufacturing workcell for study of open architecture object-oriented integration of mechatronic systems," *Chin. J. Mech. Eng.*, vol. 15, no. 2, pp. 149–152, 2002.
[56] J. Pannu, X. D. Hu, and H. H. Cheng, "Retrofitting of industrial manipulators for study of open architecture integration of mechatronic systems," presented at the ASME 18th Comput. Eng. Conf.,, Atlanta, GA, 1998.
[57] D. T. D. System, *PMAC User's Manual and Software Reference: Firmware Version 1.13*. Delta Tau Data Systems, Chatsworth, CA, Dec. 1992.
[58] Mobile Agent-Based Robotic Automation. (2005). [Online]. Available: http://www.mobilec.org/apps/robots/

Stephen S. Nestinger (M'08) received the M.S. and Ph.D. degrees in mechanical and aeronautical engineering from the University of California, Davis, in 2005 and 2009, respectively.

He is an Assistant Professor in the Department of Mechanical Engineering, Worcester Polytechnic Institute, Worcester, MA, where he is also the Director of the Robotics, Automation, and Mechtronics Laboratory. His current research interests include real-time systems, software and systems integration, multi-robot systems, collaborative cooperation, mobile agent systems, intelligent embedded and mechantronic systems, and distributed, evolutionary, and adaptive control systems.

Bo Chen (M'05) received the Ph.D. degree in mechanical engineering from the University of California, Davis, in 2005.

She is currently an Assistant Professor in the Department of Mechanical Engineering-Engineering Mechanics and the Department of Electrical and Computer Engineering, Michigan Technological University, Houghton. Her research interests include artificial immune systems and pattern recognition, mobile agent and multiagent systems, sensor networks and networked embedded systems, structural health monitoring, vehicle electronics and control networks, and intelligent transportation systems. She has authored or coauthored more than 40 refereed journal and conference papers.

Dr. Chen received the Best Paper Award at the 2008 IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications. She has been an active member of the ASME, and was a Symposium Chair and Session Chair for several international conferences. She is a member of the Executive Committee of the Technical Committee on Mechatronic and Embedded Systems and Applications (MESA), ASME Design Engineering Division.

Harry H. Cheng (SM'06) received the M.S. degree in mathematics and the Ph.D. degree in mechanical engineering from the University of Illinois, Chicago, in 1986 and 1989, respectively.

He is a Professor in the Department of Mechanical and Aerospace Engineering, and Graduate Group in Computer Science, University of California, Davis, where he is also the Director of the Integration Engineering Laboratory. From 1989 to 1992, he was a Senior Engineer for robotic automation systems with the Research and Development Division, United Parcel Service. He is the founder of SoftIntegration, Inc. His current research interests include computer-aided engineering, mobile agent-based computing, intelligent mechatronic and embedded systems, and innovative teaching. He has authored or coauthored more than 150 papers in refereed journals and conference proceedings. He holds one U.S. patent. He is the author of the book *C for Engineers and Scientists: An Interpretive Approach* (McGraw-Hill, 2009). He is the original designer and implementer of an embeddable C/C++ interpreter Ch for cross-platform scripting. He participated in revision of the latest C standard called C99 through ANSI X3J11 and ISO S22/WG14 C Standard Committees and made contributions to new C99 numerical features of complex numbers, variable-length arrays, and the IEEE floating-point arithmetic, which had been implemented in his C/C++ interpreter.

Dr. Cheng is a Fellow of the American Society of Mechanical Engineers. He was the Conference Chair and the Program Chair of the IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications.