Plug-and-Play Open Architecture Integration of Mechatronic Systems for Agile Manufacturing

Harry H. Cheng
Integration Engineering Laboratory
Department of Mechanical and Aeronautical Engineering
University of California
Davis, CA 95616

Abstract

A plug-and-play open architecture language environment for integration of mechatronic systems for agile manufacturing will be presented in this paper. This new language environment is designed to be computer platform independent, mechatronic device independent, and mechatronic system independent. The language environment is a salient extension and enhancement of Unix/Windows/C computing environment, popularly used by engineers and scientists all over the world. The language environment for real-time operation of mechatronic systems is a superset of interpretive C programming language implemented as a shell. We have designed and implemented this interpretive language environment with security provisions for cross-platform network computing. The mechatronic systems under this new integration paradigm, therefore, can be remotely operated thousands of miles away from the physical site across the network. An experimental rapid prototype system has been developed to verify our principles and algorithms on open architecture integration of mechatronic system. Hardware from different vendors can be easily plugged into the prototype system and the user can readily play it without changing application programs written in the superset of C. The new integration language environment is available for downloading on the WWW at the URL address `http://iel.ucdavis.edu/CH/`.

Keywords

Open architecture, system integration, mechatronics, superset of C, network computing.

1 Introduction

Agile manufacturing is characterized by its ability to allow a rapid response to continuously changing customer requirements and markets. The speed with which new products are developed and delivered to market often is the major determinant of competitive success. While U.S. manufacturers recognize the importance of speed and flexibility of agile manufacturing, many are impeded in efforts to become more agile producers by peculiarities in manufacturing software and incompatibilities among software applications, which necessitate costly time-consuming customized system integration efforts. The importance of open-architecture manufacturing is well-recognized by researchers [31] as well as industrial practitioners. The National Technology Roadmap for Semiconductors lists manufacturing integration tools as one of the industry's highest priorities. Similarly, the National Electronics Manufacturing Initiative cites the lack of an agreed-upon open framework for integration as a major impediment to future competitiveness. The American Electronics Association highlights system integration as one of four critical areas needing R&D resources [16].

One of major impediments to become more agile manufacturers for many industries is the lack of open software environment for rapid integration of mechatronic devices that include stepper motors, DC and AC motors, laser and vision sensors, force and torque sensors, tactile sensors, and pneumatic grippers. These mechatronic devices are essential components of many manufacturing systems such as robot manipulators, CNC machines, and automatic assembly systems. Computerized integration of these mechatronic devices is one of major efforts in development of agile manufacturing and automation systems [14]. Typically, these mechatronic devices are integrated with PCbus or VMEbus-based distributed computing systems in the C programming language. For different type of applications, user interfaces to machinery are different, even if the underlying mecha-

*published in *Open Archietcture Control Systems and Standards*, Frederick M. Proctor, Editor, Proc. SPIE 2912, pp. 136-145, 1997.

tronic devices and operation principles are the same. Significant recoding is often neeeded in response to design changes and new product lines, which implies more cost and longer delivery time.

Difficult vertical information integration throughout organizations is another major barrier to become more agile manufacturers for many industries [28]. The timely flow of information including real-time manufacturing data within and between enterprises is critical to agile manufacturing business. Currently, the costly effort is required to implement and integrate information systems that share real-time manufacturing data throughout organizations. Typically, factory-floor information systems, which focus on the operation of mechatronic systems and on the control of processes, communicate neither directly nor regularly with front office information systems, which are used for design, process planning, quality control, accounting, forecasting, and other resource planning activities. As a result, upstream information systems are unaware of important manufacturing details such as the availability of appropriate tools; records of past process; maintenance schedule; or the status of work in progress. Middle-level information systems, called Manufacturing Execution Systems (MES), bridge this critical information gap between the upstream and downstream systems. MES track and manage all aspects of a job on the factory floor in near real-time. MES play essential supervisory and monitoring roles that link all levels of manufacturing and business operations. For example, they identify bottlenecks and material shortages on the shop floor, and they provides near real-time performance of manufacturing systems along with comparison with the past performance and projected results.

However, MES are difficult to implement and integrate, and often, even more difficult to modify and upgrade. Only a small fraction of manufacturers can afford the cost of installing and maintain an integrated MES solution. Even these highly automated plants and factories struggle to overcome difficulties in adapting or reconfiguring production operations to accommodate design changes and new product lines. The National Electronics Manufacturing Initiative includes MES integration among the most important infrastructural issues that must be solved in order for U.S. electronic manufacturers to succeed in the global competition [16].

2 A New Integration Language Environment

We have developed a new integration language environment called C^H for enterprise-level system integration [10]. The C^H integration language environment is developed for plug-and-play open architecture horizontal integration of mechatronic devices for operation of mission critical real-time mechatronic systems such as robotic manipulators, CNC machines, and automatic assembly systems. C^H is also developed for vertical integration between real-time machineries and non-real-time front office and engineering information systems. C^H is a superset of C [18, 20]. Therefore, experienced C programmers can readily use the system without retraining. We have added the IEEE 754 standard for binary floating-point arithmetic to C and extended C with complex and dual data types in entire complex and dual domains, intrinsic functions, nested functions, computational arrays, arrays of adjustable index, arrays of variable length, fortran style arrays [6,7]. Some features implemented in C^H are considered by the ANSI and ISO C Standard Committees as new extensions to C. By adding thes new features to C, C^H is a very high-level language (VHLL). Many application programs can be written in C^H with a fraction of C code. We have used C^H as a teaching and learning tool for instructional improvement at both undergraduate and graduate levels [8]. C^H has been designed and implemented with security features for cross-platform network computing. C^H applets can be downloaded on demand through the network dynamically and executed in the local design and manufacturing systems on the fly [3, 4].

C^H is a salient enhancement and extension of the Unix/Windows/C computing environment with high-level features for system integration in agile manufacturing. The C^H integration language environment is *open and integrable.* C^H retains C's low-level features such as pointer for interfacing hardware. It can integrate hardware and software across vendors' product lines. It is *computer platform independent, mechatronic device independent, and manufacturing system independent.* Deterministic real-time operation is one of critical characteristics of most mechatronic system [29]. Unlike Java [1], which is not suitable for real-time applications because of non-deterministic garbage collection, C^H is designed for both real-time and non-real-time applications. Currently, it runs in Unix engineering workstations, Linux, Windows 95, Windows NT, and real-time LynxOS. C^H is computer-architecture neutral. It can virtually be ported to any computer platform ranging from hand-held electronic devices to supercomputers. The $\mathrm{C^H}$ integration language environment is *reconfigurable.* It can be reconfigured for control of different manufacturing systems to allow for rapid response to design changes and new product lines. The $\mathrm{C^H}$ integration language environment is *adaptive.* Manufacturing systems integrated using the $\mathrm{C^H}$ language environment can be modified on-line in real-time, through its built-in network interface such as Berkeley sockets and dynamic applets, and reconfigure themselves to automatically adjust to changing work loads, conditions, and requirements. The $\mathrm{C^H}$ integration language environment is *modular and scalable.* System integration in $\mathrm{C^H}$ is accomplished not by writing large programs starting from scratch. Instead, they are combined by relatively small components. These components are small and concentrate on simple tasks so that they are easy to build, understand, describe, and maintain.

How the $\mathrm{C^H}$ language environment can be used for open architecture integration of mechatronic systems will be described in sections below.

3 Computer Platform Independent Integration

The $\mathrm{C^H}$ integration language environment is designed to be computer platform independent. 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. For example, the program compilation presents a serious problem for real-time robot manipulation. For a real-time system, the external environment may be different at each execution, so the testing scenario may not be repeatable. During debugging and testing, it is impractical to restart a robot program from the very beginning every time a change is made or a problem is diagnosed [2, 14, 22, 27, 30]. Also, for adaptive manufacturing systems, dynamic applets will be downloaded from the network and executed on the fly. These applets cannot be binary executables for reasons of security and platform independence. Therefore, $\mathrm{C^H}$ is implemented as a superset of C command interpreter. The user interface of the $\mathrm{C^H}$ language environment is consistent across different computer platforms and mechatronic systems, regardless of the underlying operating systems, whether they are Unix, Windows, or real-time OS.

In command interpretive mode, $\mathrm{C^H}$ is similar to the C-shell [19]. Both $\mathrm{C^H}$ and C shells have their local shell variables besides the environmental variables. Table 1 gives a brief syntax comparison between $\mathrm{C^H}$ shell and C shell. The semantics of $\mathrm{C^H}$ commands listed in Table 1 are the same as those of the corresponding commands of C shell. It should be pointed out that identifiers starting with underscores are reserved for implementation use of the C language. Many system variables in $\mathrm{C^H}$ are implemented with leading underscores as shown in Table 1, without breaking the existing C code.

Unlike the so-called C shell, $\mathrm{C^H}$ is a superset of C. It is much more convenient to program in $\mathrm{C^H}$ shell than in C shell. Executable programs in binary executables and command scripts of other shells can be included in a $\mathrm{C^H}$ program. A C program can be executed without compilation in a $\mathrm{C^H}$ language environment. The command-line argument interface in $\mathrm{C^H}$ is C compatible. C programs are called *command files* or simply *commands* in $\mathrm{C^H}$. A file is identified as a $\mathrm{C^H}$ program if it has read/write permission and starts with one of the following tokens:

1. Comment symbols `/*` or `//`
2. A type specifier, type qualifier, or storage-class specifier.
3. Symbol `#` followed by a preprocessor directive.
4. Symbol `#` followed by `!/bin/ch`, `!/bin/rch`, or `!/bin/sch`
5. Identifier `main`.
6. Dot '.', which is used for execution of the program in current shell.

In a $\mathrm{C^H}$ programming environment, a command file can be executed without compilation. The system variable `_extc` of string type contains file extension of commands. The default value of variable `_extc` is `".ch .c"` in Unix and `".ch .exe .com .bat .cmd .c .ksh"` in Windows 95 and Windows NT.

The $\mathrm{C^H}$ language environment can recognize other shell scripts and programs. To be recognized by other shells and programs such as for WWW Common Gateway Interface, a $\mathrm{C^H}$ program shall start with

```
#!/bin/ch
```

followed by command line options such as `-R` for restricted shell, -S for safe shell, and `-f` for fast start up. There shall be no space before the pound sign `#`. A program without function main() or starting with `#!/bin/ch` is called a *script*. It is treated in the same manner as a command. Inside a script , system variables `_argc` and `_argv`, or shell variables `argc` and `argv` when header file `shell.h` is included, can be used for command line interface. These two command line interface variables are available even for a command file.

Table 1: Selected syntax comparisons between C$^{\mathrm{H}}$ and C shells.

C$^{\mathrm{H}}$ shell	C shell
_home= "/home/harry"	set home = "/home/harry"
_path = "/usr/bin /bin"	set path = (/usr/bin /bin)
umask(022)	umask 022
putenv("DISPLAY=remotehost:0.0")	setenv DISPLAY remotehost:0.0
printf("%s\n", getenv("PATH"))	echo $PATH
remenv("PATH")	unsetenv PATH
remvar("path")	unset path
int i = 90	set i =90
_host = `hostname`	set host = `hostname`
alias("rm", "rm -i")	alias rm 'rm -r'
unalias("rm")	unalias rm
ls ~ * $_path > output	ls ~ * $path > output
ls | tar -cf tarfile	ls | tar -cf tarfile
/bin/ls ../*	/bin/ls ../*
_histsize = 100	history = 100
history	history
$1 -agl	!1 -agl
$3	!3
vi `grep -l "str1 str2" *.c`	vi `grep -l "str1 str2" *.c`
more .chshrc .chlogin .chlogout	more .cshrc .login .logout

A C$^{\mathrm{H}}$ program can be divided into many separate files. Each file consists of many related functions at the top level that are accessible to any part of a program. A file that contains more than one function is usually suffixed with `.ch` to identify itself as part of a C$^{\mathrm{H}}$ program. Besides command files and script files, there are *function files* in C$^{\mathrm{H}}$. A *function file* in C$^{\mathrm{H}}$ is a program started with a function definition. A function file shall be readable. The extension of a function file is specified by the system variable `_extf` of string type. The default value of the system variable `_extf` is `".ch .ff .c"`. The names of the function file and function definition inside the function file shall be the same. The functions defined using function files are treated as if they were the system built-in functions in C$^{\mathrm{H}}$. For example, if program `addition.c` contains the following statements,

```
/* function file for adding two integers */
  int addition(int a, b) {
    int c;
    c = a + b;
    return c;
  }
```

Function addition() can be invoked automatically to add two integers. It is suggested that, inside a function file, there is only one function definition which may nest many local functions. In a program that invokes the function addition() from function file can be prototyped as

```
extern  int addition(int a, b);
```

This prototype for a function from a function file is optional in the program.

Command, script, and functions can invoke themselves recursively. They are the basic building blocks for integration of mechatronic systems. C$^{\mathrm{H}}$ is a congenial programming environment, in which users can extend the system beyond the capabilities of its original design. Like shell programming, integration software for operation of mechatronic systems and general tasks in C$^{\mathrm{H}}$ are created not by writing large programs starting from scratch. Instead, they are combined by relatively small components. These components are small and concentrate on simple tasks so that they are easy to build, understand, describe and maintain. A set of task-level commands and function files developed by experienced researchers and engineers can be readily used by novice users and factory personnel.

4 Mechatronic Device Independent Integration

A mechatronic system typically consists of mechatronic devices that include stepper motors, DC and

AC motors, laser and vision sensors, force and torque sensors, tactile sensors, and pneumatic grippers, etc. An open architecture integration language environment will enable integration of mechatronic devices from different vendors for consistent interfaces external to machinery. The mechatronic device independence of the C^H language environment can be achieved using a set of function files. For example, the base address of the servo control board on the VME bus can be obtained by function `servo_board_address()`. For a mechatronic system using servo control board from different vendors, only the code inside `servo_board_address()` needs to be modified. With this base address, registers for encoder positions and others can be easily accessed using pointers. Similarly, the base address for a VME acquisition board of a force/torque sensor can be obtained by function `force_board_address()`. The forces and torques applied to the end-effector of a robot manipulator can be obtained from registers with offsets from this base address.

For time critical communication, device drivers can be used to hide the mechatronic devices from application programmers [25]. In IEEE POSIX conforming systems, three kinds of files: ordinary files, directories, and special files including device drivers are handled by the same system calls through the kernel. I/O devices such as terminals, disks, and printers are handled through its I/O device drivers. For open architecture system integration, mechatronic device drivers can be created for consistent interface of digital I/O, analog I/O, servo control, vision sensor, force sensor cross different vendors' product lines. For fast interface to low-level mechatronic devices, besides all standard C functions defined in `stdio.h`, the following additional intrinsic functions are available in C^H: `open()`, `close()`, `read()`, `write()`, `ioctl()`. These system functions in C^H are in compliance with the IEEE POSIX standard. Function `int open(char* device, int mode)` initializes `device` for input, output or both, depending on `mode`. Function `int close(int fd)` breaks the connection between the file descriptor returned from function `open()` and the device. Functions `int read(int fd, char *buf, unsigned nbytes)` and `int write(int fd, char *buf, unsigned nbytes)` are for input and output to a device driver. Function `int ioctl(int fd, int cmd, int arg)` can be used to control how a mechatronic device is accessed.

A set of device drivers have been developed for servo control and force/torque sensing. The device drivers for servo control are located in `/dev/ch/servo` which is symbolically linked to the directory `/dev/ch/pmac` for interface with servo control boards from Delta Tau Data Systems. For interface with servo controllers from other vendors, say Galil, `/dev/ch/servo` can be symbolically linked to `/dev/ch/galil` where mechatronic-device-dependent device drivers would be located. For example, a device driver called `encoder1` for encoder value of motor 1 is located in the directory `/dev/ch/servo`. The encoder value for motor 1 of a robotic system can be obtained by the following C^H program

```
#include <fcntl.h>
main() {
  int n, fd;
  float encoder1;
  fd=open("/dev/ch/servo/encoder1",O_RDONLY);
  n=read(fd,(char*)&encoder1,sizeof(float));
  close(fd);
}
```

Although different servo controllers may be used in different robotic systems, the above interpretive C^H program remains the same for obtaining the encoder value of the first motor of robotic systems. For better performance, the above code can even be compiled using a native C compiler. As described in the previous section, a function file can also be created for consistent user interface in obtaining the encoder value. The function can be defined as `float encoder(int num)`. The input argument of the function is the encoder number and the value of the encoder is returned. For illustrative purpose, the robust error checking code is not included in the above program.

5 Mechatronic System Independent Integration

A typical production line often consists of several mechatronic systems. Integration of different mechatronic systems in a production line can be significantly simplified if the same language environment is used for operation of these different mechatronic systems. An open architecture integration language environment should be mechatronic system independent so that integration programs and functions are portable and can be readily used for applications of different mechatronic systems, which will relieve programmers of the task of learning different programming languages for different mechatronic systems. The idea of using the same language for operation of different mechatronic systems is hardly new. For example, MCL [24] extends the APT language [17], designed nearly 40 years ago for CNC part programming, to encompass robot motion control functions. However, the awkward APT syntax and its underlying design are unlikely to gain wide acceptance outside of the CNC machine community.

In our system, mechatronic system independent open architecture integration is achieved by merging mechatronic system related linguistic features to the C^H language environment in the spirit of C. Some compromises have to be made in integrating mechatronic features into the general computing language environment. For example, system variable `_home` refers to a user's home directory shown in Table 1. Global variable `_home1` indicates the home position of the default first mechatronic system in a integration language environment. Shell command `mv` can be used to move files in the system whereas function `move()` can move actuators of multiple mechatronic systems at the programmer's discretion.

The difference of mechatronic components from different vendors is handled by the low-level device drivers as described in the previous section. These low-level device drivers are axis-based and they are not suitable for motion coordination of mechatronic systems with significantly coupled kinematics and dynamics such as robot manipulators [5, 13]. The open architecture integration of mechatronic systems should be much higher level than low-level programming of device drivers. Like `float.h` in C for hiding characteristics of floating-point numbers of different computers, the difference of mechatronic systems such as robots with different number of joints is treated by modifying header files, such as `robot1.h` and `robot2.h` with different DH and inertia parameters, and system-dependent modules such as `inverseKinematics.c` for inverse kinematics [13]. The configuration of the software structure for open architecture integration of mechatronic systems is shown in Figure 1. For different mechatronic systems, only *device drivers, header files, and system-dependent modules* need to be modified, but high-level integration functions and programs remain the same. Like user interface of C shell in different Unix engineering workstations, the user interface in the C^H integration language environment is consistent across different computer platforms and mechatronic systems. With properly installed mechatronic components and device drivers, a mechatronic system be be readily played when the C^H language environment is plugged in.

For example, a C^H program for control of a Puma 560 in joint space of a robotic system with multiple robots shown in Figure 3 is given as follows:

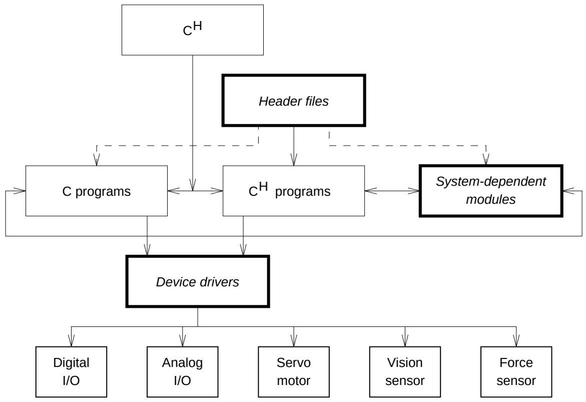

Figure 1: The software configuration for open architecture integration of mechatronic systems.

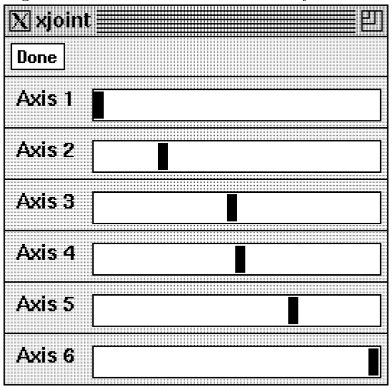

Figure 2: The graphical user interface for control of a robot in joint space.

```
#include <robot1.h>
main() {
  setupRobot1();
  xjoint 1 6
}
```

where C^H function `setupRobot1()` initializes a Puma 560 in the system with proper control parameters and calibration. Command `xjoint` is a Motif X application program for graphical control of motors. The command line arguments of `xjoint` pass the necessary information to joint level control of robot manipulators. The first argument of the command is the number of the beginning joint and the second argument is the number of joints. The output of the above program is shown in Figure 2. By dragging the cursor on an axis shown in the computer screen, a joint of the Puma 560 will move accordingly.

6 Integration of Mechatronic Systems with Other Information Systems

Because C^H can run in almost any computer systems within an enterprise, mechatronic systems operated under the C^H language environment are flexible and can be seamlessly integrated vertically with other information systems such as Web-based CAD systems [4, 12, 15]. The built-in networking facilities such as Berkeley sockets in C^H significantly enhance the integrability and adaptability of mechatronic systems.

For enterprise-level information integration, a real-time Web server can be setup in a mechatronic system under the C^H language environment. So real-time information of a mechatronic system can be made available to other information systems. The common gateway interface (CGI) is a standard for external programs to interface with the Web server [21, 11]. The C^H programs can be readily used for common gateway interface to the Web server and mechatronic system [26]. For security reason, a CGI program is typically written in restricted C^H shell.

If the `-R` flag is present when the C^H language environment is invoked as `ch -R` , the C^H shell is a restricted shell. The macro `__RCH__` is predefined with value 1 for a restricted shell. The execution environments of a restricted shell are more controlled than those of the regular shell. The actions of `ch -R` are identical to those of ch, except that the following are disallowed:

- Changing directory by cd and chdir.
- Specifying path or command names containing character `'/'` in input command, command statement, and command substitution.
- Specifying path or command names containing characters of `'/'` or `'\'` in a dot command statement.
- Specifying path with first character of `'/'` or `'\'` in `file` of `#include<file>`.
- `#include "file"` is treated as `#include<file>`
- Redirecting output `(>, >|, <>, and >>)`
- Intrinsic function `system()`.
- Using system variables `_path, _fpath, _ipath, _user, _home, _cwd, _shell, _host` as lvalue (The values of these system variables after execution of .rchshrc or .schshrc are kept for internal use. To the restricted shell user, these system variables have value of NULL).
- Using shell commands `parse` and `run` in interactive mode.
- Specifying path or command names containing ; (will be Ok in future)
- Specifying path or command names containing | (will be Ok in future).

The restrictions above are enforced after .rchshrc is interpreted. If C^H is invoked with both options -R and -f, option -f is suppressed. When a command to be executed is found to be a C^H program, the restricted shell invokes ch -R to execute it. If a C^H program is invoked with shell identification #!/bin/ch without option -R or -S, the restricted shell invokes ch to execute it. Thus, it is possible to provide to the end-user C^H programs that have access to the full power of the regular shell, while imposing a limited number of commands; this scheme assumes that the end-user does not have write and execute permissions in a directory containing commands. Therefore, the writer of the .rchshrc has complete control over user actions, by performing guaranteed setup actions and leaving the user in an appropriate directory (probably not the login directory). A default directory CHHOME/rbin/Rbin has been setup for putting commands that can be safely invoked by restricted shell.

On the other hand, mechatronic systems integrated in the C^H language environment can communicate and obtain information from other information systems such as a CAD system through network. Mechatronic systems can be modified on-line in real-time through applets dynamically loaded from the network and reconfigure themselves to automatically adjust to changing work loads, conditions, and requirements. The dynamic applets are typically handled using a safe shell.

If the `-S` flag is present when the C^H language environment is invoked as `ch -S` , the C^H shell is a safe shell. The macro `__SCH__` is predefined with value 1 for a safe shell. The execution environments of a safe shell are more controlled than those of the regular shell and restricted shell. The actions of ch -S are similar to those of the restricted shell. In addition to the restrictions imposed in a restricted shell, the following features are disallowed.

- Declaration of pointers. For example, char *str;
- Casting of pointer types. For example, `int a[10]; printf("str = %s\n", (char *)a);`
- Intrinsic functions access(), addkey(), chkey(), execv(), fopen(), fork(), getenv(). lstat(), open(), pipe(), popen(), putenv(), remkey(), remove(), rename(), socket(), socketpair(), stat(), utime().
- Using an array name as pointer in lvalue of assignment statements. For example, int a[15]; *(a+10)=12; is not allowed. But, a[10] = 12 is ok.

The restrictions above are enforced after .schshrc in the home directory is interpreted. For maximum security, a system administrator may take the ownership of `.schshrc` and change the mode of the file to readable only. If C^H is invoked with option -S, options -R and -f will be ignored. These additional restrictions are relaxed for function files located in a client. Therefore, an important safety guidance is not to use arguments from a function file as input to restricted functions directly. If an array is used as a pointer to void or pointer to char in the following functions fgets(), fread(), gets(), memcpy(), memmove(), memset(), read(), recv(), sprintf() strcat(), strcpy(), strncat(), strncpy() memory outside the array boundary will guarantee not to be contaminated. The contaminated memory by functions fscanf(), scanf(), sscanf() are set to null character to close a possible security hole. If a program try to write to the memory outside the array boundary, an error message will be generated. Since explicit pointers are not available in safe shell, these functions will be safe to use.

When a command to be executed is found to be a C^H program, the safe shell invokes ch -S to execute it. If a C^H program is invoked with shell identification #!/bin/ch without option -R or -S, the safe shell invokes ch to execute it. Thus, it is possible to provide to the end-user C^H programs that have access to the full power of the regular shell, while imposing a limited number of commands; this scheme assumes that the end-user does not have write and execute permissions in a directory containing commands. Therefore, the writer of the .schshrc has complete control over user actions, by performing guaranteed setup actions and leaving the user in an appropriate directory (probably not the login directory). A default directory CHHOME/rbin/Sbin has been setup for putting commands that can be safely invoked by safe shell.

7 A Prototype of Agile Mechatronic System

To verify our principles and algorithms of plug-and-play open architecture integration of mechatronic systems, an experimental mechatronic system with multiple industrial manipulators has been developed at the Integration Engineering Laboratory of the UC Davis. The hardware configuration of the experimental setup is shown in Figure 3. We have retrofitted the Puma 560 and IBM 7575 controllers in Figure 3. The retrofitted robot controller consists of servo controller, I/O and A/D interface boards from Delta Tau Data Systems, machine vision system from Datacube and Panasonic, force/torque sensing system from JR3. The mechatronic system is coordinated in the C^H language environment under a real-time operating system LynxOS [23] from Lynx Real-Time Systems running

Figure 3: The experimental setup for open architecture integration of mechatronic systems.

in a VMEbus based single board computer MVME167 from Motorola. There are several computer workstations, X-terminals, and PCs in the lab to drive robot manipulators. In this mechatronic system, C^H is used as login shell for operation of multiple robot manipulators. A Web server also runs in the system to communicate with other computer and information systems in the lab. We are in the process of testing of using dynamic applets to respond to external sensory information and requests from other information systems

8 Conclusions

A new open architecture integration language environment C^H for integration of mechatronic systems for agile manufacturing has been presented in this paper. C^H is a salient extension and enhancement of the most popular Unix/Windows/C computing environment. It is computer platform independent, mechatronic device independent, and mechatronic system independent. C^H is designed to be computer architecture nuetral and to run in any computing devices ranging from hand-held electronic devices to supercomputers. Mechatronic device dependent information are hidden in device drivers. Mechatronic dependent information such as DH and inertia parameters for a robotic system are stored in header files and system-dependent modules. Using this plug-and-play integration language environment, a new mechatronic system can be quickly developed in the same manner as a Unix or Windows software is installed in a computer.

As a superset of C, C^H retains C's low-level features for interface to hardware. With its new features, C^H is a very high-level language (VHLL) and to be as easy to use as Basic. Because C^H is implemented as a shell, a large collection of existing utilities and C programs can be readily used for integration of mechatronic systems. Based upon the concept of shell programming, our integration environment is open, modular, and scalable. Software for integration of mechatronic systems are created not by writing large programs starting from scratch. Instead, they are combined by relatively small components. Functions, commands, and scripts are the basic building blocks for integration of mechatronic systems, These programs can be created and executed dynamically based on the external sensory information, which can be critical to the intelligence of a mechatronic system. Built-in networking features such as Berkeley sockets and restricted/safe C^H shell provide both flexibility and security for remote operation of mechatronic systems [9]. C^H is designed especially suitable for Web-based client/server computing. C^H programs can be used for common gateway interface in the Web server and as dynamic applets executed through the Web browser. Since C^H can be used for operation of real-time mechatronic systems and tasks in front office and engineering department, enterprise-level upstream and downstream information systems can be seamlessly integrated under the C^H integration paradigm. This new integration language environment is available for downloading on the WWW [3].

Acknowledgements

This work was supported in part by the National Science Foundation under Grant DMI-9309207 and DMI-9622588, by the University of California, Davis through faculty research grant program, by the California Department of Transportation, and by industrial affiliates of the Integration Engineering Laboratory, BSDI, Datacube, Delta Tau Data Systems, HP, Intel, JR3, Lynx Real-Time Systems, Motorola, Panasonic Broadcast & Television Systems, and Unmanned Solutions.

References

1. Arnold, K. and Gosling, J., *The Java Programming Language*, Reading, MA, Addison-Wesley Pub. Co., 1996.
2. Blaha, J. R., Lamoureux, J. P., and McKee, K. E., Currently Available Robot Programming Languages, *Control and Programming in Advanced Manufacturing* (edited by Rathmill, K.), Springer-Verlag, New York, 1988, pp. 67-83.
3. Cheng, H. H., The C^H Language Environment Homepage, `http://iel.ucdavis.edu/CH/`.
4. Cheng, H. H., CAD/CAM on the Internet and Distance Learning, `http://iel.ucdavis.edu/design/`.
5. Cheng, H. H., Real-Time Manipulation of a Hybrid Serial-and-Parallel-Driven Redundant Industrial Manipulator, *ASME Journal of Dynamic Systems, Measurement, and Control*, Vol. 116, December, 1994, pp. 687-701.
6. Cheng, H. H., Extending C and FORTRAN for Design Automation, *ASME Trans., Journal of Mechanical Design*, Vol. 117, No. 3, 1995, pp. 390-395.
7. Cheng, H. H., Extending C with Arrays of Variable Length, *Computer Standards and Interfaces*, Vol. 17, 1995, pp. 375-406.
8. Cheng, H. H., Pedagogically Effective Programming Environment for Teaching Mechanism Design, *Computer Applications in Engineering Education*, Vol. 2, No. 1, 1994, pp. 23-39.

9. Cheng, H. H., Hypercollaborative Human-Machine Interaction through the World Wide Web, *Proc. 4th National Conference on Applied Mechanisms and Robotics,* Vol. II, Cincinnati, OH, Dec. 10-13, 1995, pp. AMR-50-01/04.
10. Cheng, H. H., *Quick Reference Guide to the* C^{H} *Language Environment Version 0.1, Release CH96.10.14*, Integration Engineering Laboratory, University of California, Davis, CA, October 14, 1996.
11. Cheng, H. H., CGI Programming in C, *C/C++ Users Journal*, Vol. 14, No. 11, November, 1996, pp. 17-21.
12. Cheng, H. H., Network Computing and its Applications in Design and Manufacturing, *Proc. NSF Design and Manufacturing Grantees Conference*, Seattle, WA, January 7-10, 1997.
13. Cheng, H. H., and Gupta, K. C., A Study of Robot Inverse Kinematics Based upon the Solution of Differential Equations, *Journal of Robotic Systems*, Vol. 8, No. 2, April 1991, pp. 159-175.
14. Cheng, H. H., and Penkar, R., Stacking Irregular-Sized Packages by a Robot Manipulator, *IEEE Robotics and Automation Magazine*, Vol. 2, No. 4, December, 1995, pp. 12-20.
15. Cutkosky, M. R., Tenenbaum, J. M., and Clicksman, J., Madefast: Collaborative Engineering over the Internet, *Communications of the ACM*, Vol. 39, No. 9, September, 1996, pp. 78-87.
16. Goldstein, B. L. M, Daum, M., Knutilla, A., Ray, S., and Tucker, G., Technologies for the Integration of Manufacturing Applications, Advanced Technology Program, May 1995.
17. IIT Research Institute, *APT Part Programming Language*, McGraw-Hill Book, Co, New York, 1967.
18. ISO/IEC, *Programming Languages - C*, 9899:1990E, ISO, Geneva, Switzerland.
19. Joy, W., *An Introduction to the C Shell*, Department of Electrical Engineering and Computer Science, University of California at Berkeley, 1980.
20. Kernighan, B. W. and Ritchie, D. M., *The C Programming Language*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1st edition, 1978; 2nd edition, 1988.
21. Berners-Lee, T., et al., The World-Wide Web, *Comm. ACM*, Vol. 37, No. 8, Aug. 1994, pp. 76-82.
22. Lozano-Perez, T., Robot Programming, *Proc. of the IEEE*, vol. 71, No. 7, July, 1983, pp. 821-841.
23. Lynx Real-Time Systems, Inc., *LynxOS Application User's Guide*, release 2.2.1, Los Gatos, CA, December, 1993.
24. McDonnel Douglas, Inc., *Robotic System for Aerospace Batch Manufacturing*, Feb. 1980.
25. Nicolson, E. J., Standardizing I/O for Mechatronic Systems (SIOMS) Using Real Time UNIX Device Drivers, *Proc. IEEE International Conference on Robotics and Automation*, vol. 3, 1994, pp. 3489-3494.
26. Paulos, E., and Canny, J., A World Wide Web Telerobotic Remote Environment Browser, *World Wide Web Journal*, 1996.
27. Shimano, B., VAL: An Industrial Robot Programming and Control System, *Proc. IRIA Sem. on Languages and Methods of Programming Industrial Robots*, Rocquencourt, France, June, 1979, pp.47-59.
28. Shofield, S., and Wright, P. K., A Review of Equipment Integration in a Flexible Manufacturing System, *Manufacturing Science and Engineering*, ASME OED-vol. 64, pp. 93-100.
29. Stewart, D. B., Schmitz, D. E., Khosla, P. K., The Chmira II Real-Time Operating System for Advanced Sensor-Based Control Applications, *IEEE Trans. on Systems, Man, and Cybernetics*, vol 22, No. 6, Nov., 1992, pp. 1282-1295.
30. Unimation, Inc., *UNIVAL Robot Controller, User's Guide to VAL II*, version 4.0, 1987.
31. Wright, P. K., and Greenfeld, I., Open Architecture Manufacturing: the Impact of Open-System Computers on Self-Sustaining Machinery and the Machine Tool Industry, *Proc. Manufacturing International' 90*, ASME vol. 2, March 1990, pp. 41-47.