Mechatronics Vol. 4, No. 7,
pp. 713-735, 1994

EMULATION OF MODULAR MANUFACTURING MACHINES USING CAD MODELLING

C. D. WRIGHT and K. CASE

Department of Manufacturing Engineering, Loughborough University of Technology,
Loughborough LE11 3TU, U.K.

Abstract - Designing, testing and debugging a machine control system which controls mechatronic hardware elements can be a complex, time consuming and costly procedure. It is often extremely difficult for the system builder to envisage in advance the effects of small changes to the control system logic, with potentially dangerous results if the hardware consists of heavy machinery. It is also rare that a system builder will arrive at a working prototype on the first attempt and discovering the reasons for incorrect operation without a suitable means of comprehending the problems can be an arduous task. This paper describes an approach which supports the designing, testing and debugging of modular manufacturing machines using 3D graphical models of the machine hardware. The paper emphasises the underlying methodology of the approach, which involves collecting timing data from the executing control system under development, then emulating the operation of the machine by using this data to drive a graphical model of the hardware. The term "emulation" is used to mean modelling using data captured from the real machine as opposed to "simulation" which synthesises data.

The work builds on previous research at the MSI Research Institute concerned with the control of modular machines. Two new extensions to this work are described here, which form the basis of the emulation capability. The first is the addition of the ability to execute the control system without the mechatronic hardware elements present whilst still retaining the operational behaviour of the application. The second is the mechanism for collecting the run-time data which defines these operational characteristics, to drive the machine emulation. The features of the custom 3D modeller are presented and its use for machine emulation is described. An example of a real control system under development is given to illustrate the complete process.

The research objectives of the work described here are concerned with the fundamental problems designers encounter when trying to prototype the control systems of modular machines. The research has shown that the ability to execute the control system with or without the mechatronic hardware elements present can be a considerable advantage if supported by a CAD-based emulation system.

1. INTRODUCTION

The use of computer controlled machines is now commonplace in industry although the potential for these reprogrammable systems to reduce manufacturing life cycle costs has yet to be fully realised [1]. There is an urgent need to radically improve the effectiveness of the machine and associated control system design/build process *and* to enable efficient modification of both as requirements change. The commercial impact of such improvements would be profound, seen as more rapid machine design-build-installation-setup, improved product quality, enhanced economics via smaller batches, minimum work in progress, reduced scrap and ultimately reduced overall costs.

Computer controlled machines consist of suitable configurations and combinations of

mechatronic (mechanical and control system) elements with a behaviour determined by specific application code. The inability of current methods to efficiently cater for the graphical visualisation of these mechatronic elements and the design of associated application specific software, particularly as systems increase in complexity, is seen in spiraling applications costs [2]. The application software is difficult to design, maintain and modify as it is not easy to foresee or predict the effect on machine operation and any consequent problems.

One traditional solution is to run a simulation (graphically or otherwise) of a proposed system or changes to an existing system [3-12]. The simulation can use its own internal pseudo timebase, or can be run in real-time based on mathematical models [13-15]. However, coding for the real control system processes is often done after the simulation phase and the process internals frequently bear little relation to the mechanisms used to drive the simulation. Recent research is investigating the use of petri-nets to emulate the dynamic behaviour of real systems [16] but to date the work has focused on non-time-critical systems. Additionally, both research and commercial simulation and modelling systems for computer controlled machines focus primarily on robot-based systems, offering little support for modular machines [17].

The approach outlined in this paper is fundamentally different and offers a means of overcoming these difficulties, specifically with regard to modular machines. The concern here is to provide graphical support for design evaluation rather than initiation (validation of initial system designs, proving upgrades and modifications, reconfiguration of existing systems), which enables designers to rapidly try out “what-if” control scenarios in a safe (non-destructive) environment using the real control processes. Building on the outputs of previous research work at the MSI Research Institute, extensions to the established Universal Machine Control (UMC) control system architecture [1, 18, 19] have been implemented which allow execution of the control system without the mechatronic hardware elements present. This is used in conjunction with graphical tools and models of the hardware for examining and verifying machine operation. The benefits of traditional simulations are retained (case of on-screen visualisation, access to data, safety, repeatability, support for assessing new ideas [3, 8-11]) but in conjunction with the real control system processes.

The sequence information used to drive the model elements is collected directly from the executing control processes and hence reflects the true timing interrelation-ships of the processes. This is particularly important when the control system consists of a large number of concurrent interacting processes, where the timing characteristics and overheads of the control software cannot be adequately represented in a simulation. Execution of the sequence information on the machine models is termed “emulation” and can be used to examine such issues as the interaction characteristics of the control processes, particularly with respect to communication with the external hardware elements [20] (time delays, overheads), resource deadlock and conflict [21], physical positioning of machine hardware elements and interference/collision potential.

The four main requirements of the approach are as follows.

(1) A control system which can be executed without the mechatronic hardware elements and

still retain the operational behaviour of the application. This gives us our "safe" environment for rapidly trying “what-if” scenarios on the application under development.

(2) A control system which provides a high level of data accessibility, allowing sequence/timing/sample data to be collected with minimal impact on the application.

(3) A tool for constructing graphical models of both the mechatronic elements within the machine and its operating environment.

(4) A mechanism for importing the run-time data into the modelling tool for graphical examination and machine emulation (driving the machine model).

The work described in this paper covers all of the above requirements. Extensions to earlier work [22, 23] have provided solutions to (1) and (2) above. Requirements (3) and (4) have been satisfied by building a custom modelling tool using the functionality of a commercially available solid modeller [8].

The modelling tool (the "Manipulator Modeller") implements a means of building up a model workspace from a library of modular building blocks (kinematic models of mechatronic elements). Performing machine emulations by driving the models with run-time data is used purely for on-screen visualisation of machine behaviour. Functionality within the modeller provides for examination of sequence information, sample data, event timing, input/output values and positioning of axes; in this way a complete picture of the operation of the machine can be built up. The modelling tool is not concerned with the modelling of mechanisms or control algorithms, nor is it directly concerned with robot modelling. The modular control philosophy adopted [22] endorses the provision of multi-axis capability by utilising (for example) several single degree of freedom actuators. This can be reflected in the modelling tool by constructing kinematic chains from individual actuator models.

An important feature of this work is that the mechatronic hardware elements can be incrementally added or removed. If they are not present (usually the situation when the control system is initially under test) their logical operation is emulated by software processes within the control system. This enables problems with the overall machine logic to be detected at an early stage without costly damage to hardware. If the mechatronic elements are present, the techniques can be used to monitor and fine-tune the implementation of the control system as it nears completion. For example, samples can be collected from the hardware (e.g. position, velocity, acceleration, input/output values), which can be viewed from within the modelling tool to compare actual performance with the performance obtained by running the system without the real hardware. This also allows examination of the overheads incurred when communicating with the hardware.

The discussion begins by presenting a brief overview of the existing control system on which the body of the work has been built. This is followed by a description of the additions to the control system and an outline of how run-time data is collected. The features and operation of the modelling tool are then described, including running of a machine emulation. Finally the complete methodology is illustrated using an example of a real control system under development.

2. OVERVIEW OF THE MODULAR MACHINE CONTROL METHODOLOGY

This section is included for background information, to give an understanding of the characteristics of the modular control philosophy necessary for the new work described in sections 3 onwards.

2.1. Building blocks

A UMC control system is comprised of a set of concurrent software processes which conform to a reference architecture, together with off-line software tools for configuration and management of the run-time system [1, 19, 22]. The methodology relates entirely to software modules, but provides an architecture for integrating proprietary mechatronic products and presenting a unified view of these products to the run-time software processes. These "controlled" products are termed "external devices" as they are not part of the collection of software processes which form the control system. A "machine" includes the external devices and consists of concurrent control system processes and associated data structures, mechanisms and protocols for inter-process co-ordination and communication, off-line software tools for control system configuration and management, and the external devices which are to be controlled (e.g. motion controllers. I/O controllers, sensors, software models). Figure 1 illustrates the run-time architecture of a machine.

Proprietary design products and tools can be used in conjunction with the control system, including graphical modelling tools for machine emulation and evaluation

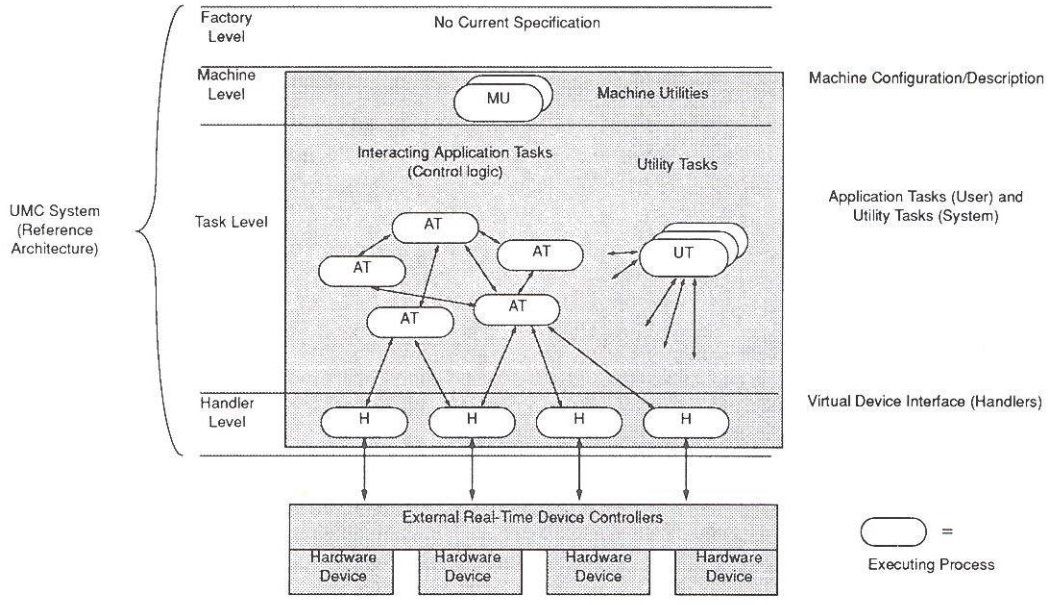

Fig. 1. The run-rime architecture of a UMC based machine.

(discussed in this paper), tools for motion design [24] and application programming/logic design [25-27].

2.2. Advantages of the methodology

The modularity inherent within the methodology is of fundamental importance as it permits reconfiguration and extension [1, 19, 22]. Reuse of common software building blocks enables control systems to be constructed in a unified and coherent manner, leading to improved system maintenance and upgrade capabilities.

Complex control problems can often be decomposed into a set of software processes which each perform relatively simple functions, making them easier to maintain and test. This can result in significant time and cost savings in comparison with the upgrading or modification of proprietary systems [28]. It also permits graphical tools to be employed for designing the control system logic which until recently has only been possible in very few vendor specific systems [29, 30]. Further detailed information on theoretical and implementation issues of the control system can be found in [22, 23].

3. CONTROL SYSTEM EXTENSIONS

Two extensions to the control system are described. Both of these are necessary to support the graphical modelling tools described later, but provide valuable additions to the capability of the control system irrespective of whether the graphical tools are used.

The first extension enables the control system application to be executed with or without the external mechatronic hardware elements. There are several reasons why it is desirable to be able to do this. Frequently, the control system software develops at a different rate to the hardware. Indeed, the general machine hardware requirements will usually be known (e.g. the number and type of input/output lines, sensors, actuator combinations and parameters such as maximum required velocity/acceleration, limits of motion, etc.) before decisions regarding suppliers and distributors of specific hardware have been finalised. Execution without the hardware implies that control application development can be divorced from hardware related problems, making the application development environment much safer, less costly (not open to expensive mistakes with the hardware) and allowing the machine builder to rapidly experiment with new control strategies.

The second extension is the addition of the ability to log data from the executing control system. The data relates to the logical operation of the control application and its interaction with the mechatronic hardware. Again, there are many reasons why logging control system data is desirable. The data can be used for fault finding, status and error reporting, performance monitoring (e.g. process and device response times, examining the timing interactions and overheads incurred as control processes are added and removed and those due to communication with the hardware elements [20, 21]), sampling sensors and sampling of hardware elements (input/output values, actuator

position/velocity/acceleration), or simply as a trace of the operation of the machine.

3.1. The existing external device interface

The lowest level control system processes provide a "virtual device interface" between the internal unified control system and the outside world [1, 19, 22]. These processes, called handlers, typically interface to external device controllers and allow translation between device specific and control system generic messages. External devices include single and multi-axis motion controllers, ports and binary I/O lines and software models of real devices, but may be of almost any type and degree of intelligence. Each external device interfaces to the control system processes via its own handler (Fig. 2) through which all device specific communications pass.

3.2. Extensions for execution without external device hardware

To execute the control system without the external devices, the handler for each device has to be replaced with one which incorporates the functionality of the device and can respond to device-related messages from the application processes. The replacement handler is termed an "emulation" handler, as it emulates the behaviour of the external device in software, in terms of how the device is seen from the higher level application processes. An emulation handler implements mechanisms for calculating the duration of requested operations, performing time delays, keeping track of the status of the imaginary external device and supplying sample information on request (current motion parameters, current input/output values). It does not implement mathematical models to mirror the dynamic behaviour of the external device.

To the application processes within the control system, emulation handlers appear identical to real handlers, as they respond to requests in the same logical manner. The application logic remains unchanged. Figure 3 shows the functional relationship between an emulation handler and a real handler (a motion control handler). The diagram also illustrates the data logging method, outlined in the next section.

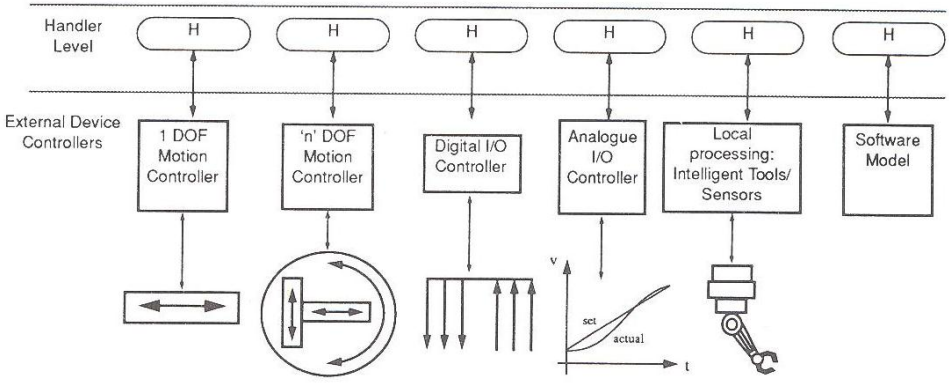

Fig. 2. Different external devices.

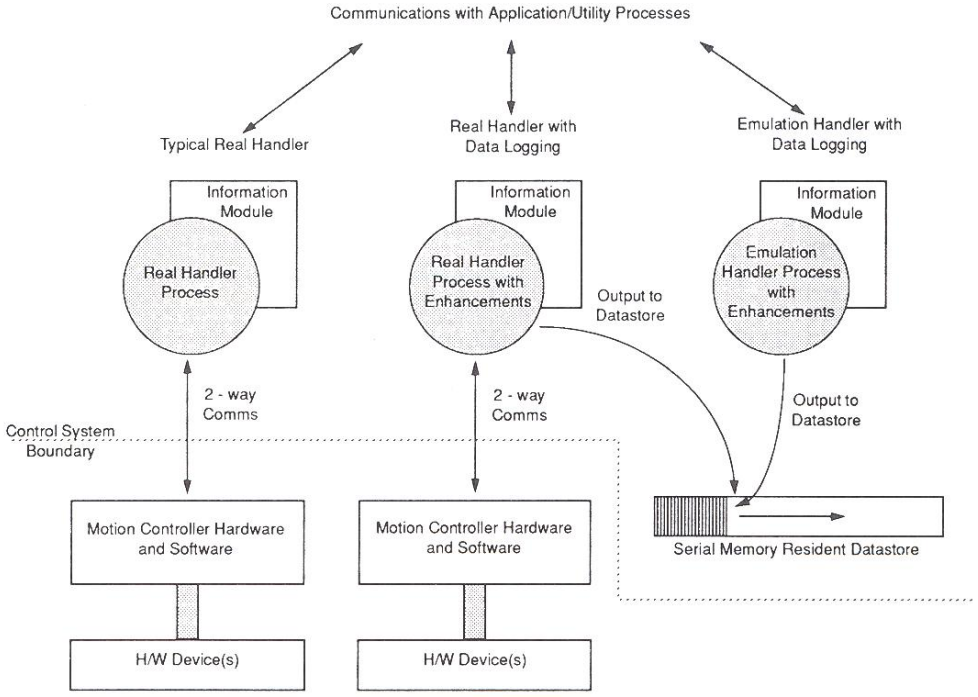

Fig. 3. Real and emulation handlers.

3.3. Extensions for logging of run-time data

Data visibility is inherent in the methodology behind the control system. Application processes communicate with handler processes by using common data areas called "information modules" which can be read at any time by any process executing in the host environment [1, 19, 22, 23].

The functionality of handlers (both real and emulation) has been extended to include capabilities to log this data to a central serial datastore. This is illustrated in Fig. 3. When a handler receives an action request from an application process, information relating to the request is appended to the datastore. All handlers within a particular control system write to the same datastore and access it using a locking mechanism to prevent resource conflict. The datastore is memory resident for fast access, to ensure that the locking and writing mechanisms have negligible influence on the operation of the control system. Action requests which can be logged are given in Table 1. Typically a particular external device will be capable of responding to a subset of these.

Figure 4 shows a section of a typical data log. Due to the real-time behaviour of the system it is not possible to predict the order in which the blocks of data appear. Each block begins with the system time (in system ticks) when requested action is serviced by the handler and is followed by the data passed to the handler as part of the requested action. The log can be thought of as a run-time audit trail; the timing

Table 1. Handler action requests

Action	Description
ABORT	Stop axis motion as fast as possible
ENDMAP	Stop mapping a slave axis to a master axis
ENTERMAP	Download a motion mapping to a motion controller
ENTERPRF	Download a motion profile to a motion controller
EXEMAP	Start mapping a slave axis to a master axis
EXEPRF	Execute a motion profile on an axis
HOME	Home an axis
JOG	Execute a constant velocity move on an axis
MOVEPOS	Execute a triangular move on an axis
MOVETRAP	Execute a trapezoidal move on an axis
READANALIN	Read an analogue input line
READDIGIN	Read a digital input line/channe
SETANALOUT	Set an analogue output line
SETDIGOUT	Set a digital output line/channel
SERVOING ON/OFF	Tum a motion controller servo on/off
STOP	Stop axis motion
UPDATE	Sample an external device
WAITFIN	Wait for a handler operation to complete

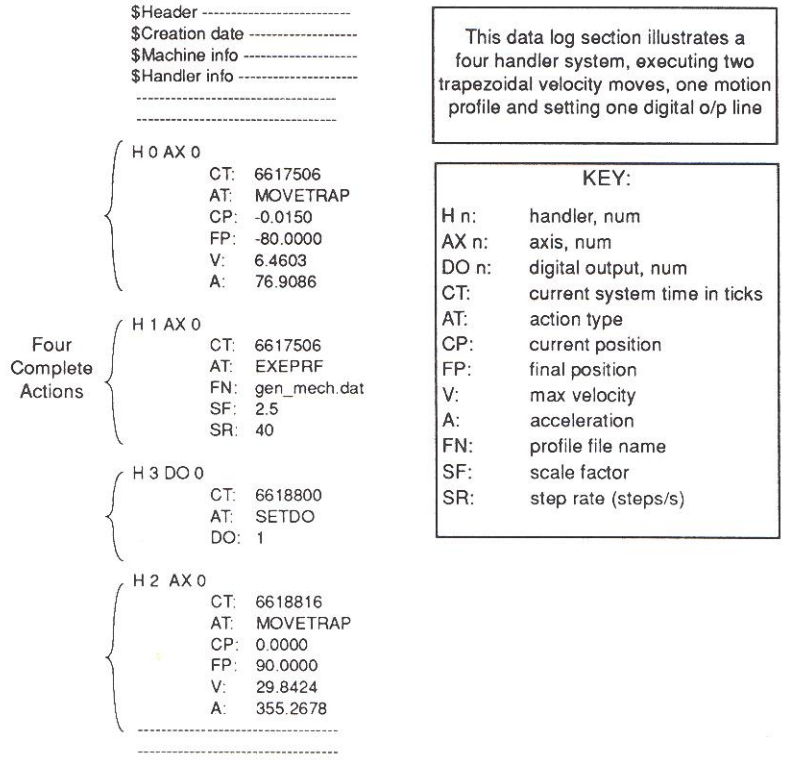

Fig. 4. Section of a data log.

of the data blocks it contains is accurate to the resolution of the system tick (typically of the order of 10 ms [20, 31]).

The control software can incorporate custom motion profiles and mappings designed using a proprietary motion modelling package [24]. These consist of either position/time or position/position information for sets of axes. This information is stored in a separate file which is referenced in the data log when execution of the profile or map is initiated. An example of this is shown in Fig. 4.

3.4. Application development issues

There is no restriction on the combination of real and emulation handlers in a particular machine. The control system can be run with all emulation handlers, all real handlers (and real hardware) or a mixture of both. Hence, incremental application development is supported: initial testing and debugging of prototype application processes typically involves a completely emulated system with data logging. As the development of the machine progresses emulation handlers are successively replaced with real handlers. With all handlers logging data, the log contains a complete description of the application operation.

The same principles apply to data logging. Any combination of handlers can be logging data at a particular time. Logging can be turned on or off as the control system is running-useful for diagnostics and maintenance. If the control system is highly time-critical it may also be desirable to leave logging on permanently in the final application, to incorporate any time distortion caused by the logging mechanisms.

In a real handler the time taken to perform a particular operation is determined by the capabilities of the external device and the type of action requested. In contrast, an emulation handler must explicitly calculate this time. Clearly, the calculations may differ to some extent depending on how well the emulation handler reflects the actual behaviour of the real external device it replaces. However, the primary concern is in examining the logical operation of the complete machine, rather than the detailed performance characteristics of individual external devices.

4. THE MANIPULATOR MODELLER

4.1. Requirements and capabilities

The inherent advantages of graphically representing a problem to aid comprehension by the user are well accepted [30, 32, 33]. The fundamental requirement here is for a general purpose graphical modelling tool for examining the control system data logs discussed previously. To achieve this the tool needs to provide a means of constructing graphical models of both the mechatronic elements within a machine and its operating environment, together with a mechanism for importing data logs and driving the model elements to perform machine emulation.

Many commercially available modelling packages provide facilities for modelling industry standard robots and supply a library of robot configurations with the package [8, 12]. The modeller simulates the desired robot operation and produces code (e.g. VAL 2) for driving the real robot. Other commercial packages are concerned more with discrete

event simulation [9-11] for applications such as materials handling, job shop throughput, etc. [3].

The approach adopted here differs from these in two important aspects. Firstly, this research goes further than simply modelling robots and adopts a more general approach by providing facilities for modelling actuator chains of any desired type, length and orientation. These consist of models of real or desired actuators which can be assembled, edited, reconfigured and saved to a library as required, giving the system builder much greater flexibility and freedom. Secondly, as the control system is not being simulated the internals of proprietary modelling packages for driving the animations are of no concern to us. We do not need to implement a programming mechanism for keeping track of event occurrence as the timing interrelations of the control data have already been set by the executing control system. What is required is a means of replaying our control actions against an arbitrary timebase and graphically displaying the timing interrelationships of the data.

Because of these issues we are not directly concerned with robot modelling, mechanism modelling or control algorithms. Robot and mechanism models *can* be constructed, but from modular distributed building blocks. The information required to animate the models still comes from the imported control system data log: the initial design of the control algorithms to drive them would be undertaken elsewhere.

The Manipulator Modeller is based on a commercially available modelling package [8] but with additional custom facilities for working with modular building blocks. The primary reason for using a commercial package was to remove the considerable development overheads incurred when implementing the display drivers and underlying data structures of a graphical modelling system.

4.2. Outline of operation

The machine designer is provided with facilities to create, orient, edit and compare manipulator models, to add geometrical shapes from a partbox, to save the manipulator models to a library and to drive the models with loaded data logs. As new manipulator models are created they can be saved to the library for future use. The user can then browse through the library and choose the most appropriate manipulator building blocks as the complete machine model is constructed. In addition, as part of the proprietary modelling package a full range of standard 3D modelling facilities is available for interacting with the model workspace [8].

The Manipulator Modeller enables the machine designer to quickly build up a representation of a machine using serially linked chains of up to three degrees of freedom (DOFs). Each DOF can be either revolute (R) or prismatic (P) and consists of a white base part and a colour coded moving part. When a DOF is created it is given a set of default kinematic parameters consisting of maximum position, minimum position, home position, maximum acceleration and maximum velocity. The parameters can then be individually edited, listed and compared for each DOF in the model. Dynamic properties are not currently modelled. Several example manipulator models are shown in Fig. 5.

Commercially available single actuator or multiple actuator serial mechanisms can be modelled using any combination of 1, 2 or 3 DOF building blocks. The user is not restricted to a specific configuration and the solution to a particular design problem can

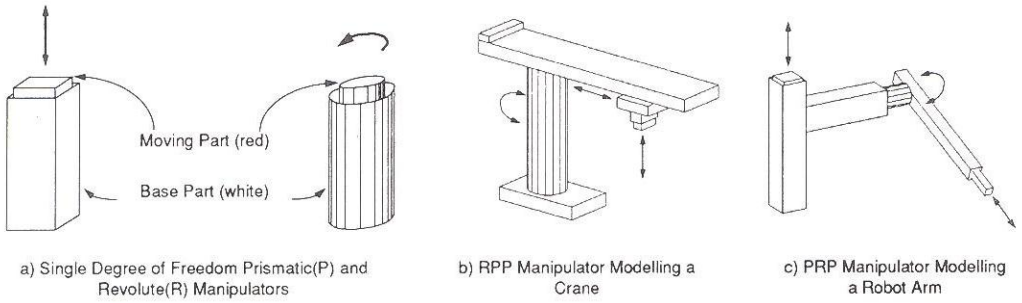

Fig. 5. Examples of 1 and 3 DOF manipulators.

often be realised with several different combinations of building blocks. The whole workspace can be constructed from single DOF actuators if required. This follows the control system philosophy of providing the machine builder with the mechanisms for building machines from modular components [1, 19, 22] rather than complex proprietary building blocks (e.g. robots or robot arms), resulting in lower system cost, enhanced maintainability and reusability. Models of several commercially available actuators are shown in the worked example, Fig. 9.

Each single DOF of a manipulator model consists of a hierarchical data structure built from several different types of data block. Figure 6 shows the data structure for the first DOF of the simple crane illustrated in Fig. 5b. The data structure employs a hierarchical ring mechanism commonly used for boundary representation modellers [34].

When a manipulator model is created the user simply enters a name for the whole manipulator. The data blocks in each DOF are automatically created and interconnected as illustrated in Fig. 6. The moving part of a DOF can then be moved relative to the base part by using the spatial connecting block, in this case CRANE_X*n*. Any positional changes written to the homogeneous transformation matrix in this block will affect the moving part geometry and any DOFs further down the serial chain. By definition, all movements within a DOF take place about the local Z axis of the spatial connecting block (Fig. 7).

DOF motion is achieved using a function which writes directly to the spatial block transformation matrix, to perform either translation along the local Z axis (prismatic DOFs) or rotation about the local Z axis (revolute DOFs). This is the basic driving function used by the routines which perform machine emulation. The calculations to perform rotation and translation about the spatial connecting block axis operate by concatenating homogeneous transformation matrices [35].

4.3. Machine emulation

Data logs such as the one illustrated in Fig. 4 can be loaded directly into the Manipulator Modeller. Any profiles or maps referenced in the log are loaded at the same time. A memory block is dynamically allocated for each distinct action in the data log and all of the blocks created from one log (including optional profiles/maps)

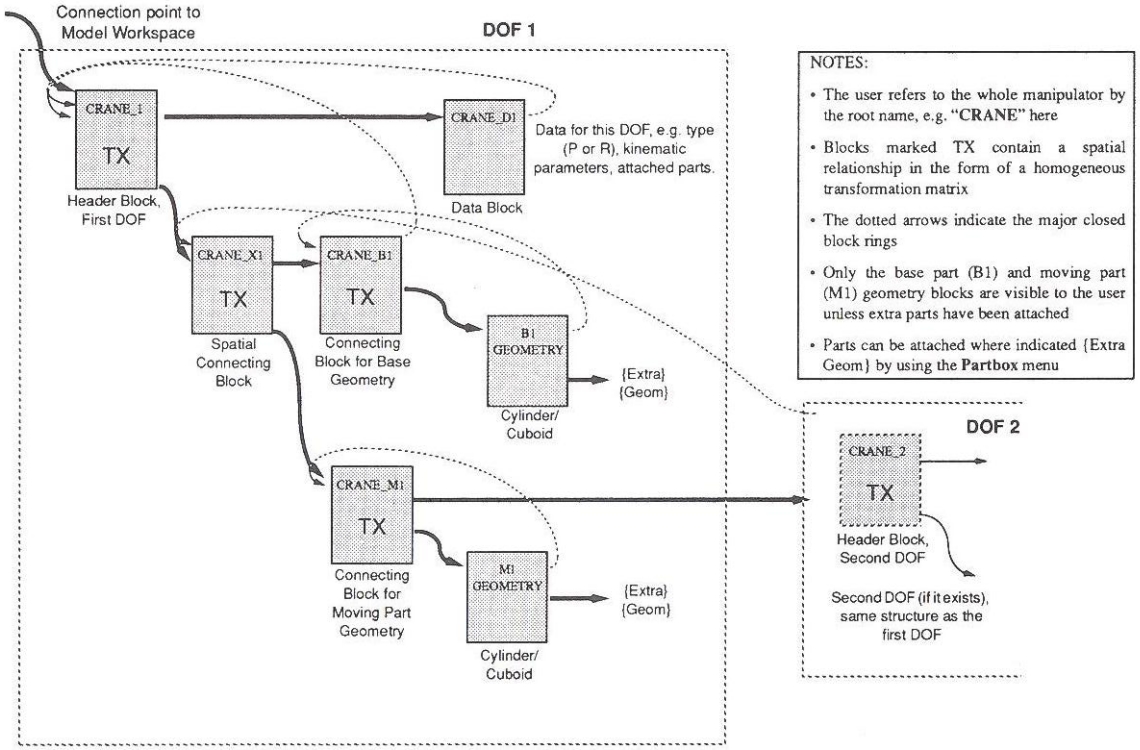

Fig. 6. Data structure for each single DOF.

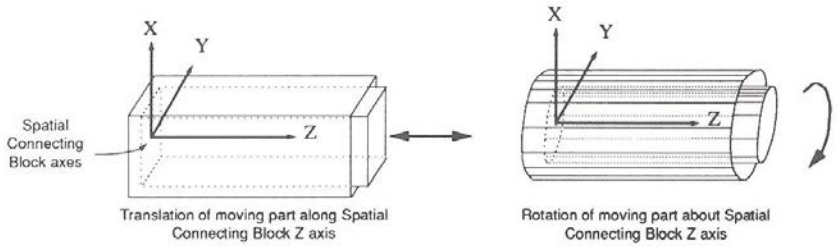

Fig. 7. DOF motion.

are linked to form an "emulation process". Duration times for the complete process and individual actions are calculated while the log is being loaded. The starting times for all actions are normalised relative to the first action which occurs, designated as time zero. In Fig. 4 the first action is a move which occurs on handler SH 0 at time 6,617,506 ticks: this becomes time zero after loading. Multiple emulation processes can be created by loading further data logs, useful for comparing similar operating sequences on the same machine, such as the results of fine-tuning the application software.

To execute the machine emulation, actions in the emulation process are assigned to the required model DOFs. These are performed on a “per-handler” basis (e.g. all actions from handler 1, axis 2 drive model element GANTRY, DOF 2). By changing the assignments any process or part process can be executed on any set of manipulator DOFs, assuming they are of the correct type (P or R). This allows multiple models to be developed simultaneously. The emulation can be executed with any chosen display time interval between successive frames, in single step mode or in continuous mode. At each time interval the process data blocks are scanned for active actions, positions of actuators are calculated, the appropriate matrix transformations are performed (see the previous section) and the model display is updated.

There are many diagnostic options available as part of the emulation. The process can be stepped through either forwards or backwards and the timestep can be changed at any stage. The user can then zoom in on features of the model for close examination of operations, particularly valuable when checking for possible collisions or verifying the position of elements within the workspace. Steps can be in time increments or single operations (e.g. Goto the start of trapezoidal move number 5, handler 3, axis 0). Prior to running an emulation the parameters for the motion operations on a particular handler can be checked against the assigned DOF, to verify that they lie within the kinematic motion limits of the DOF. Checking can be included to report on the status of individual DOFs during process execution (e.g. position relative to specified maximum and minimum, percentage of maximum velocity/acceleration). Data from a single handler or group of handlers can be executed alone, for example to verify the operation of a single manipulator.

As part of the toolset for running the emulation, a graphical timing tool allows the complete process to be displayed as a timing chart. This shows displacement, velocity, acceleration, input/output values and sample information for all of the actions on each

handler in the process. Samples can be overlaid (if the log contains them) in a variety of display formats such as crosses, squares, or continuously connected. The beginning of each action can optionally be marked with a vertical bar and labelled with the action type. Plot detail such as scaling, labelling and marking of maximum and minimum points can be added if required. Individual handler plots are automatic- ally scaled and labelled with maximum and minimum values, handler numbers and assigned DOFs in the model. Units are automatically calculated from header information in the data log. The display can also be scrolled, expanded, compressed and zoomed in/out to examine action timing relationships in detail and multiple displays can be invoked for comparison of more than one process. A vertical cursor can be positioned at the current process time and continually updated as the emulation runs.

Figure 8 shows three plots of a log which originated from a system consisting of a single real handler (with data logging) driving three axes, with the handler sampling the hardware position at regular intervals when a move was in progress. Samples are marked on the displacement plot with large crosses. The continuous curves in the plot show what the hardware was asked to do: here the vertical bar at the start of each action represents the system time at which a command was issued to the hardware.

In a traditional simulation the information for changing the ownership of model objects to enable them to be transported around the workspace (such as printed circuit boards, containers or objects being picked up in a gripper jaw) is included as part of the sequence information entered by the user. Because we are working with real data, the log does not contain this information. Items in the Manipulator Modeller workspace can be picked up or set down by adding model ownership changes to process actions, inserted after stepping to the correct place in the process. One use of this is to make I/O handler operations such as activating a gripper, pick up the object being gripped. Initial object positions can also be stored at the start of the process. The worked example in the next section requires several ownership changes to be inserted.

Both the executing emulation and the timing displays are a precise representation of the information contained in the loaded data logs. The timing accuracy of the log data is limited by the resolution of the control system tick. With this proviso, the emulation offers a means of examining detailed aspects of the control system operation in a form which is readily assimilated.

5. USING THE TOOLS - A WORKED EXAMPLE

A printed circuit board component insertion application has been developed using a modular testbed and a UMC control system [23]. The application is based on typical operations which a component insertion workcell might be required to perform. The example does not address issues relating to choice of hardware for a particular application, nor does it detail the methods employed to design the layout and hierarchy of the application tasks in the control system. The objective here is to illustrate how a developing machine can be emulated; it is assumed that strategic decisions relating to initial design issues have already been taken.

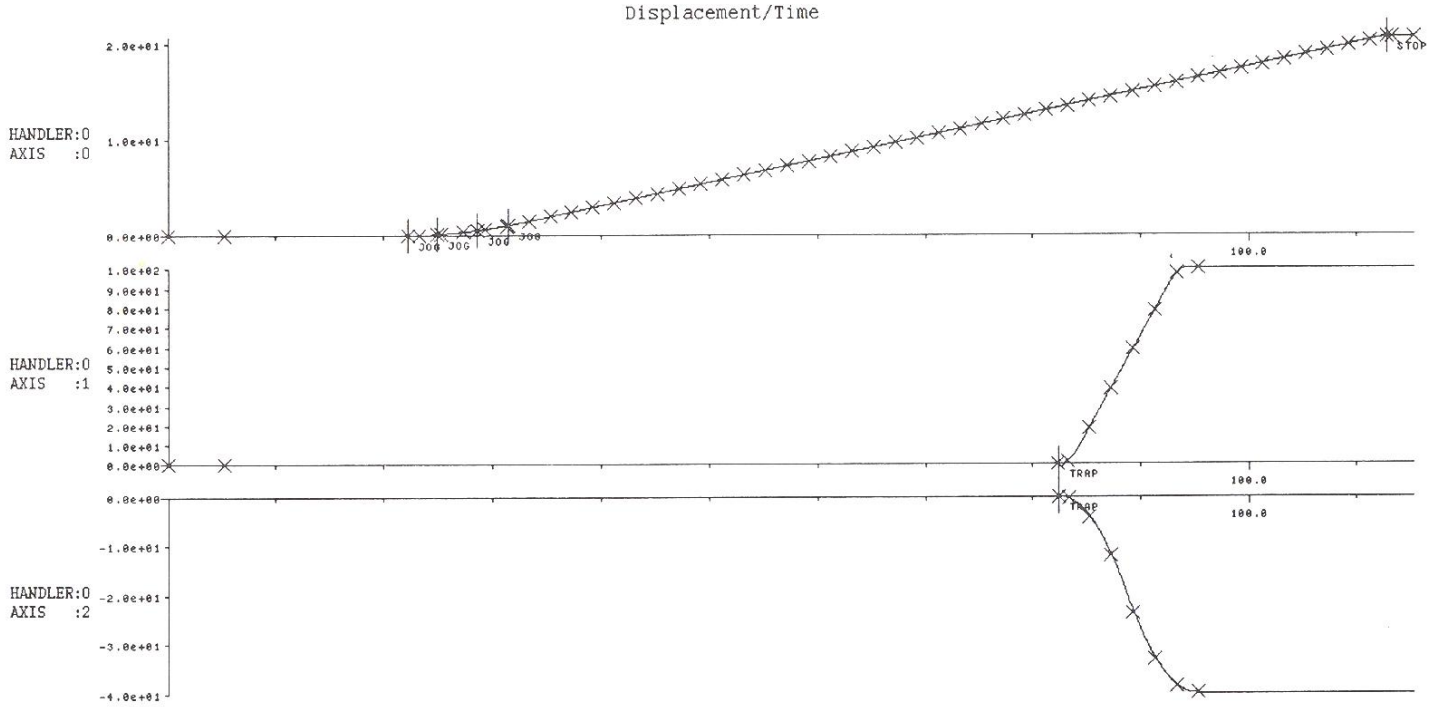

Fig. 8a. Displacement/time graphics plot, three axes.

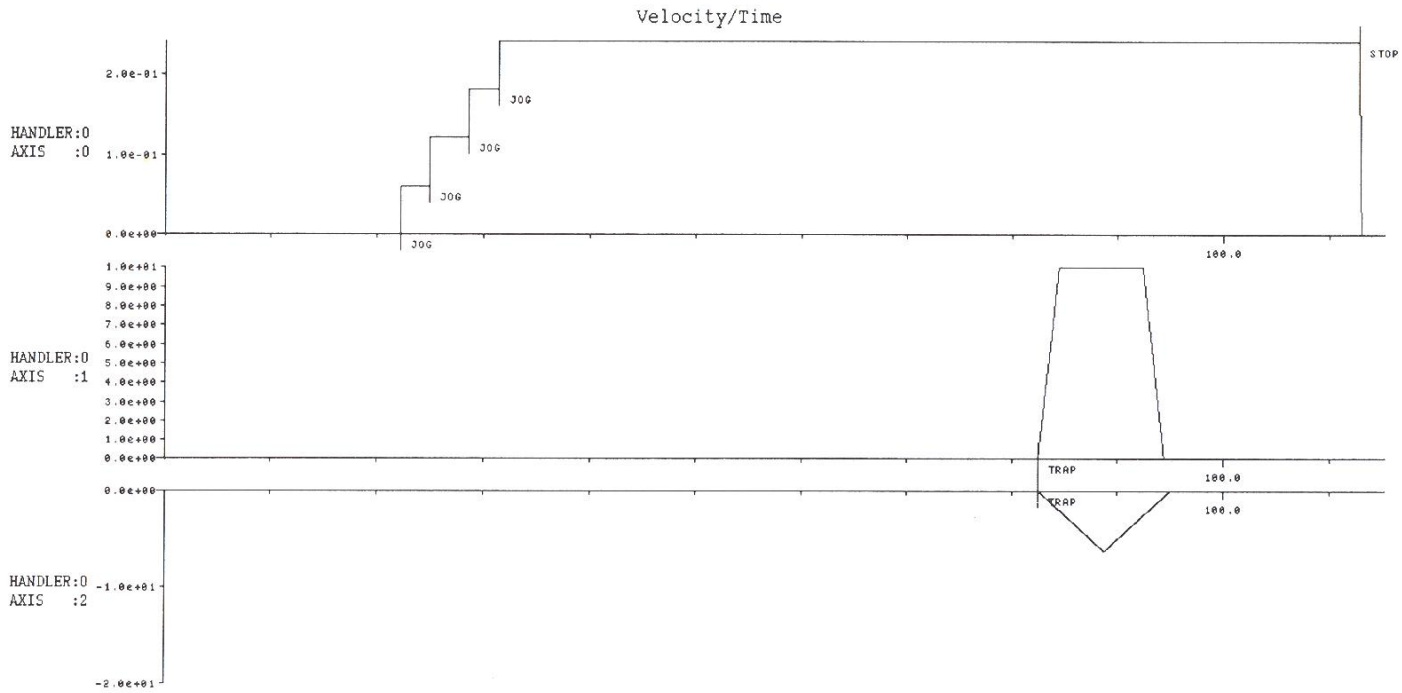

Fig. 8b. Velocity/time graphics plot, three axes.

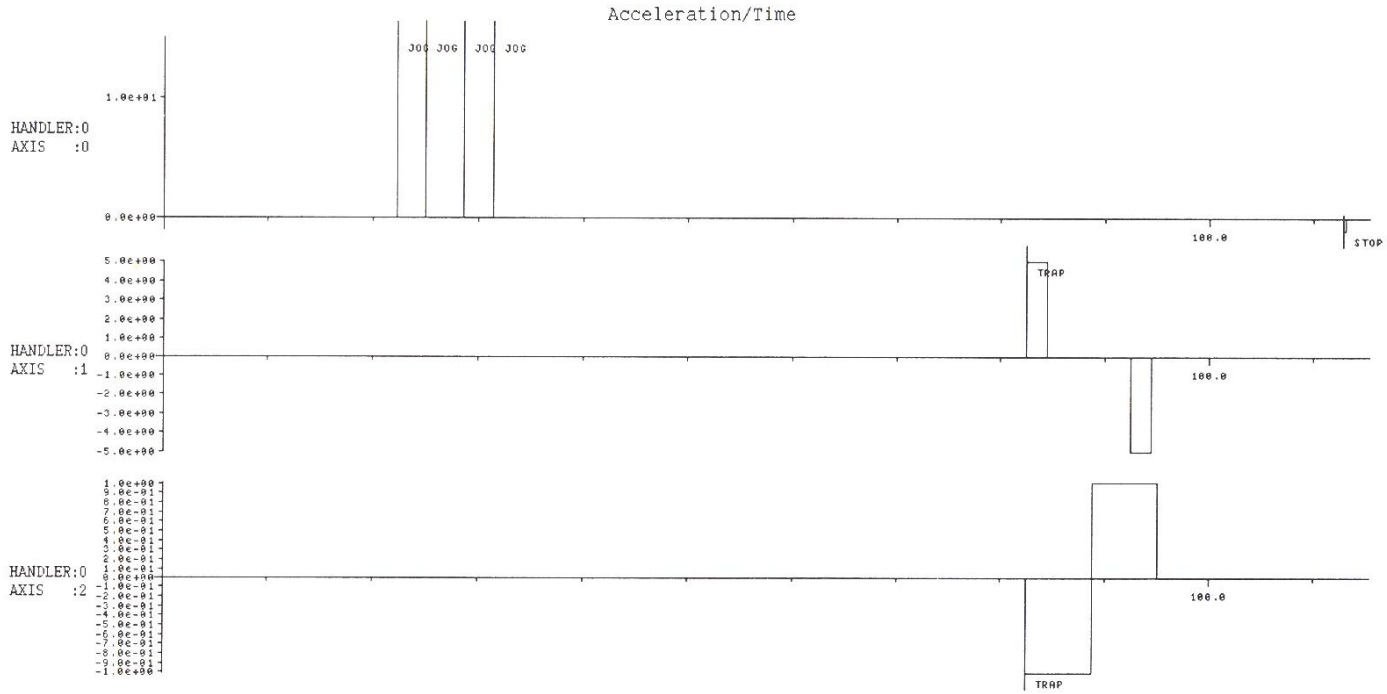

Fig. 8c. Acceleration/time graphics plot, three axes.

5.1. Building the manipulator models

Firstly, a solid model of the complete workcell must be constructed to use for the emulation. This involves either creating new manipulator models for the chosen hardware, or retrieving existing models from the manipulator library. The demonstration hardware consists of two multi-DOF manipulators and three single DOF manipulators, with a total of 10 axes of motion. Manipulator models of these are shown in Fig. 9.

The complete testbed with the models in place is shown in Fig. 10. This diagram also shows a model of the box which holds the circuit boards when they arrive in the workcell.

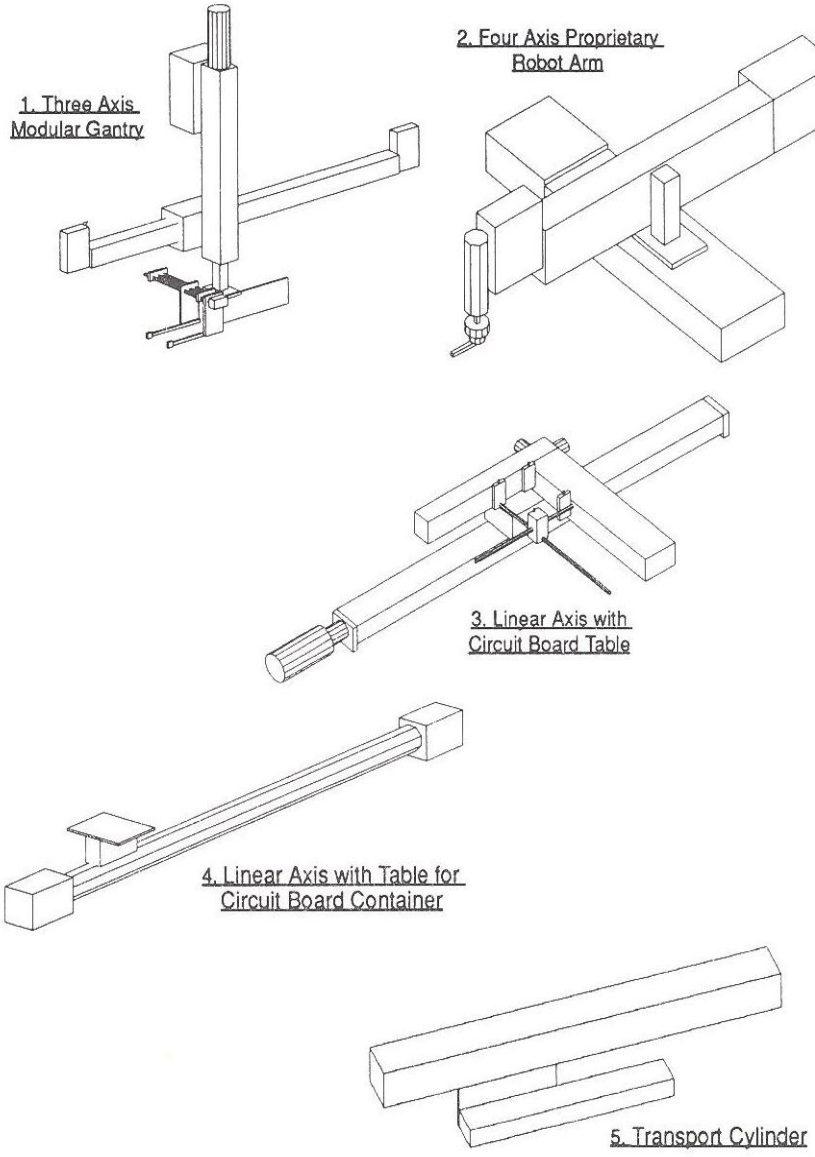

Fig. 9. Manipulator building blocks.

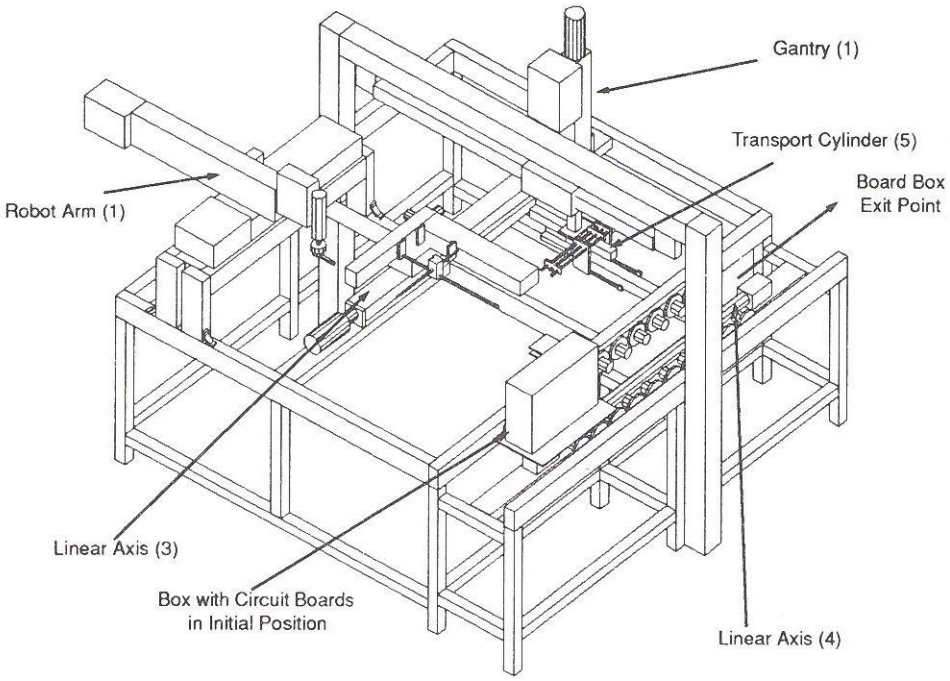

Fig. 10. Model of testbed framework with manipulators and circuit board box.

5.2. Workcell operation

A box containing printed circuit boards arrives in the workcell by means of a conveyor (not shown in the model). The box is shown in Fig. 10 in this initial position. The workcell is required to insert a component into each board, then eject the board box from the back of the cell. An outline of the application logic is as follows.

(i) The board box is picked up by the linear axis (4) and moved towards the gantry (1).
(ii) The gantry gripper moves onto a circuit board.
(iii) The transport cylinder (5) which is positioned just behind the gripper (the transport cylinder is attached to the vertical gantry axis) pulls a board from the box and slides it between the gripper jaws.
(iv) The gantry moves the board to the linear axis with circuit board table (3) and drops it onto four location pins on the table.
(v) The linear axis (3) moves the board along until it is underneath the four axis robot arm (2), which then inserts a component.
(vi) The board is returned to the board box by reversing the previous stages. The return position is different from the initial position.
(vii) The process is repeated for the next board until all boards have been processed.
(viii) The board box is ejected from the cell.

The next stage is to generate a set of application tasks to perform these required logical operations. Details of how this is achieved are given in [23].

5.3. Generating the data log

A prototype control system is now created, consisting of the application processes and a set of emulation handlers [23]. For this example 10 single axis emulation handlers are initially employed, each with data logging capabilities. The machine is executed (without hardware) and the data log is generated.

5.4. Creating and running the emulation process

The data log is loaded into the Manipulator Modeller to create an emulation process. Output from each of the 10 handlers in the process is assigned to the required DOFs in the model. Timing characteristics for the axes can be examined using the graphical process display tool. Figure 11 shows the tool with handler and DOF assignments on a displacement/time plot.

A time interval is chosen for successive frames of the emulation, which can be run continuously or in single step mode. The vertical dotted line in Fig. 11 shows the current process time. The circuit boards in the board box can be driven around the model by stepping through the process and inserting interactive ownership changes, then re-running the emulation. Sections of the process can be examined in detail by stepping through and zooming in to view specific model elements.

If the emulation highlights any errors in the application logic, the application tasks must be edited, recompiled and re-run to generate a new data log, which can be loaded alongside the original data for comparison. The procedure can be iterated as many times as desired until the operation of the emulated machine is satisfactory. The emulation handlers can then be successively replaced with real handlers (still with data logging) and the control system executed with some or all of the real hardware in place. The data log now incorporates any delays introduced by the hardware and the graphical displays can be used for examining these as the application logic is fine-tuned. When the machine operation is correct, all of the emulation handlers will have been replaced with real handlers.

6. IMPLEMENTATION ISSUES

The current implementation of UMC v3.0 runs on Motorola 68xxx family processors under the OS-9 real-time operating system [31]. The Manipulator Modeller runs on a Sun Sparcstation and is based around the GRASP Solid modeller, a commercially available boundary representation robot modelling package [8]. The Modular Systems Research Group (MSG) at the MSI Research Institute has an open version of GRASP which allows custom software to be linked to the runfile and new menu layouts to be designed and included. The standard GRASP runfile together with this suite of custom software are collectively termed the “Manipulator Modeller”. The research began in the mid-1980s and is now at an advanced stage.

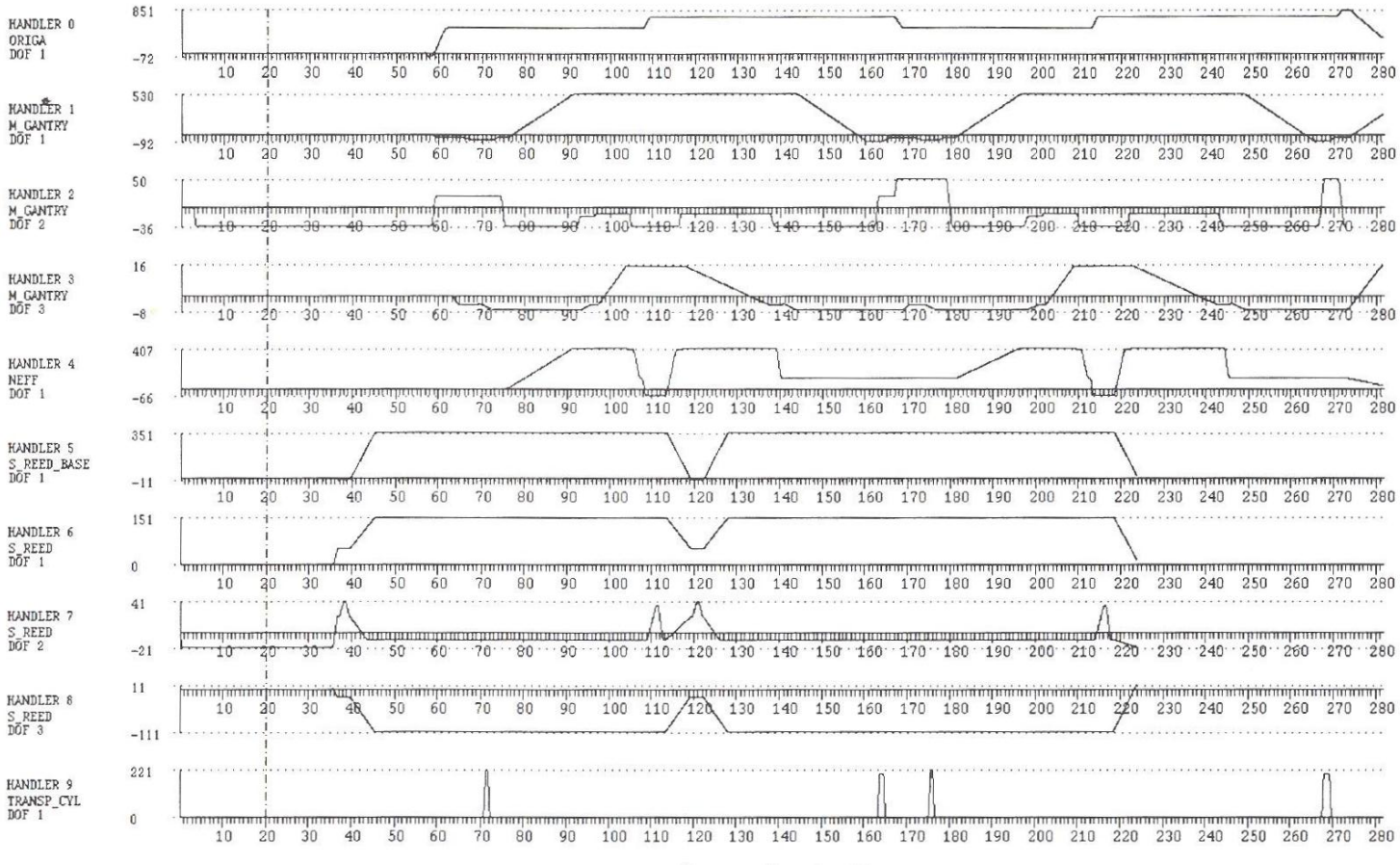

Fig. 11 Displacement/time graphics plot, 10 axes.

7. CONCLUSIONS

This paper has outlined an effective approach for debugging and testing a developing modular machine which is based on the UMC methodology. The approach uses a custom 3D modeller as an integral part of the complete design and test cycle, allowing the logical operation of the developing machine to be examined at any time. By emulating the communications with machine hardware and removing the need for the hardware to be present, the system builder can incrementally build and test machines in a safe, non-destructive environment. In contrast to traditional simulation methods data collected from the executing control system is used to drive a model emulation of the machine, ensuring that overheads imposed by the control system software are incorporated at all stages.

The real advantage of the methodology becomes apparent when the application logic of the machine needs to be altered or extended, or performance (throughput) needs to be enhanced. The effects of changes to the application logic or hardware elements can be rapidly investigated without the need to invest in expensive hardware for prototypes. The modular nature of the modelling tool allows the machine models to be reconfigured and enhanced extremely quickly, whilst the graphical timing diagrams ensure that operation of the machine is clear and easily visualised on-screen. The complete process enables strategy decisions to be taken with increased confidence.

Future work is looking at integrating the model elements into the run-time control system such that emulation handlers will drive mechatronic models directly (not off-line, as in the current situation). The intention is to make a bi-directional connection between the target run-time platform and the Manipulator Modeller, enabling the model elements to interact with the control processes in real-time. This will permit more detailed modelling of the mechatronic elements to include sensors and limit switches, capable of sending signals back to the control system. The system will then provide increased support for modelling input/output generation, fault situations and error conditions.

Acknowledgements-The research described in this paper is funded by the Applications of Computers in Manufacturing Engineering (ACME) directorate of the Science and Engineering Research Council (SERC) and by the SERC/DTI High Speed Machinery LINK initiative. The authors wish to acknowledge the important contributions to work on UMC concepts and implementation from the following members of the MSI Research Institute at Loughborough University: Prof. R. H. Weston, Dr P. R. Moore, Dr R. Harrison, A. H. Booth, A. J. Carroll and N. A. Armstrong.

REFERENCES

1. Weston R. H., Harrison R., Booth A. H. and Moore P. R., A new concept in machine control. *J. Comput. lntegr. Manufact. Syst.* **2**(2), 115-122 (1989).
2. Muir K., Stating the CASE for industrial automation. *Drives Controls* **June** 22-24 (1990).
3. Pidd M., *Computer Simulation in Management Science,* 3rd edn. John Wiley, New York (1992).
4. Springfield J., Cook G. E., Andersen K. and Fernandez K. R., ROBOSIM: a simulation package for robots. *Proceedings of the 7th International Conference of the*

ASCE, pp. 239-246, Laramie, WY, 23-26 July (1989).
5. Levas A. and Jayaraman R., WADE: an object-oriented environment for modelling and simulation of workcell applications. *IEEE Trans. Robotics Automat.* 5(3), 324-335 (1989).
6. Paul B. and Rosa J., Kinematics simulation of serial manipulators. *Int. J. Robotics Res.* **5**(2), 14-31 (1986).
7. Ravani B. and Hornick M. L., STAR: a simulation tool for automation and robotics. In *Control and Programming* in *Advanced Manufacturing*, International Trends in Manufacturing Technology, pp. 269-294, IFS, Bedford (1988).
8. BYG Systems Ltd., The GRASP solid modeller reference manual. Highfields Science Park, University Boulevard, Nottingham, U.K. (1987).
9. AutoMod 3D simulation software. AutoLogic Systems Ltd., Parkmeads, 37 Oast House Crescent, Farnham, Surrey GU9 ONP, U.K. (1993).
10. Witness, Visual interactive simulation software. AT&T ISTEL, Visual Interactive Systems Ltd., Highfield House, Headless Cross Drive, Redditch B97 SEQ, U.K. (1990).
11. The HOCUS simulation system. P-E Consulting Services Ltd., Park House, Wick Road, Egham, Surrey TW20 OHW, U.K. (1992).
12. Robot simulations. WORKSPACE v2.0-The PC based robot simulation. 21 High Bridge, Newcastle upon Tyne NEI lEW, U.K. (1991).
13. Urquhart T., Darner P. and Habibi S., Principles of real time simulation. Cambridge Control Ltd., Report for Esprit II, Project No. 2617, Report No. CONT.410.CC.Dl. (1991).
14. System Specs by Ivy Team. Available from BSO/Tasking UK Ltd., 16 Fernhill Road, Cove, Farnborough. Hants GU14 9RX U.K. (1993).
15. Sagoo J. S. and Holding D. J., The specification and design of hard real-time systems using timed and temporal petri nets. *Microprocess. Microprog.* 30, 389-396 (1990).
16. Aguiar M. W. C., Coutts I. A. and Weston R. H., Rapid prototyping of open software systems. *European Simulation Multiconference, ESM'94*, Barcelona, Spain. 1-3 June (1994).
17. Yan X. T., Graphical modelling of modular machines. Ph.D. thesis. Loughborough University of Technology, U.K. (1992).
18. Advanced architecture for universal machine control. SERC/ACME research report, Grant GR/F13492, MSI Research Institute, Loughborough University of Technology, U.K. (1992).
19. Harrison R., A generalised approach to machine control. Ph.D. Thesis, Loughborough University of Technology, U.K. (1991).
20. Dickenson W. and Breame A., Computing for real-time. *Electronics Wireless World* March, 193-196 (1994).
21. Ben-Ari M., *Principles of Concurrent Programming.* Prentice-Hall, London (1982).
22. Harrison R., Wright C. D., Carrott A. J., Booth A. H. and Armstrong N. A., UMC conceptual specification. MSI Research Institute, Loughborough University of Technology, U.K. (1992).
23. Booth A. H. and Carrott A. J., UMC version 3.0 reference manual. MSI Research Institute, Loughborough University of Technology, U.K. (1993).

24. Limacon. Camlinks and motion users manual. Meadow Farm, Horton. Malpas. Cheshire SY14 7EU, U.K. (1991).
25. Select Software Tools Ltd., Select Yourdon USER GUIDE. Idsall House, High Street, Prestbury, Cheltenham GL52 3LY, U.K. (1993).
26. Yourdon E., *Modern Structured Analysis.* Prentice-Hall, Englewood Cliffs, NJ (1990).
27. SOT by TeleLOGlC. Reflex Technology Ltd., 3 Buckingham Place, Bellfield Road, High Wycombe, Bucks HP13 SHW, U.K. (1993).
28. Glad A. S., Software engineering guide for real-time control systems development. *Sixth Control Engineering Conference,* pp. 188-199, Rosemont, IL (1987).
29. Texas Instruments, Applications productivity tool. Siemens TI Ltd., Siemens House, Eaton Bank, Varey Road, Congleton, Cheshire CW12 1PH, U.K. (1992).
30. CJ International, ISAGRAF v3.10 industrial software architecture, user's manual. 44 Rue Pre Didier, 38120 Le Fontanil, France (1990).
31. Microware Systems Corporation, The OS-9 real-time operating system, OS-9 Technical Manual. Des Moines, IA 50322 (1989).
32. Foley J. D. and Van Dam A., *Fundamentals of Interactive Computer Graphics.* Addison-Wesley, Reading. MA (1982).
33. Kramer J., Magee J. and Ng K., Graphical configuration programming. *Computer* **22**(10), 53-65 (1989).
34. Mantyla M., *An Introduction to Solid Modelling.* Computer Science Press, Rockville, MD (1988).
35. Fu K. S., Gonzalez R. C. and Lee C. S. G., *Robotics,* pp. 14-52. McGraw-Hill International, Singapore (1987).