First Humanoid, Nanotechnology, Information Technology, Communication and Control Environment and Management (HNICEM) International Conference
March 27-30, 2003, Holiday Inn, Manila, Philippines

A NEW COMPUTER AIDED DESIGN ENVIRONMENT USING INTELLIGENCE THAT IS DISTRIBUTED THROUGHOUT ADVANCED PRODUCTION MACHINES

DA SANDERS
University of Portsmouth,
Portsmouth PO1 3DJ
United Kingdom
david.sanders@port.ac.uk

TEWKESBURY GE
University of Portsmouth,
Portsmouth PO1 3DJ
United Kingdom
giles.tewkesbury@port.ac.uk

ABSTRACT
With complex designs involving many manufacturing processes, a designer may have a broad understanding of the overall process, but a limited detailed knowledge of the individual processes. New techniques to analyse design situations and to implement the results within a new user interface are presented. The new design environment focused and guided a designer in the task of designing tooling for plastic part manufacture. The work demonstrated that in the future a designer's knowledge might be collected and analysed for use in an automated system. This may allow part or all of the design process to be automated. To demonstrate this new design environment, a small production area was created that included new types of task machine and virtual task-machines. The new task machinery was successfully integrated into the production environment and automatically scheduled, configured, and programmed.

Key Words – COMPUTER, DESIGN, INTELLIGENCE, DISTRIBUTED, MACHINES.

I. INTRODUCTION

A new restricted programming language has been created to program task oriented machines. This led to a new method of integrating advanced production machinery. The method allows a design level to interface directly to a manufacturing environment. The manufacturing level advises on aspects of the production tasks, which are necessary for the design. The method offers a route towards full factory automation, and integration with a design level.

Programming approaches at a factory level are presented, and novel techniques to fully automate factories are described. The framework allows the interrogation and programming of any level within a factory hierarchy. Each level of an automated factory hierarchy has been included and a case study is described to demonstrate a successful application of the approach. The results of this application are presented and conclusions drawn.

The concept of task machines was first presented in 1992 [1]. Task machines are machines constrained for a task, but are not product dependent. A task machine would therefore be specified in terms of task rather than functional abilities, for example, a robot which can 'spray panels' rather than a robot which has 'six degrees of freedom'.

In the past only single task machines have tended to be programmed [2]. This paper reports on progress towards a method for integrating and automatically programming more than one task machine within a simple production environment created during the research.

A new method for customising virtual task machines was created by mapping general-purpose machinery programming languages to a new restricted programming language. This new technique was used to create two virtual task de-flashing machines. These individual task machines were integrated into work-cells, and then into a production environment. This was achieved by the creation of work-cell and factory co-ordinators.

Many robot-programming languages exist [3,4], and more are being developed. A programmer faced with the task of programming many different machines has to cope with various

levels of complexity, functionality, and structure found in each of the languages encountered. This variance can slow the writing and maintenance of application programmes.

A new alternative approach is to take the essential functional commands from each of the different machines and to map them to a new common restricted programming language. The commands are mapped in such a way as to provide continuity when changing from the programming of one machine to another. This continuity extends to the units and order of parameters passed to the new commands.

The approach is suited to the quick customisation and programming of machinery for applications. This makes it particularly useful in the creation of virtual task machines, enabling a task machine to be quickly configured from any general-purpose machinery.

The application of this approach to the creation of a generic functional programming environment is described. The environment was used by an experienced programmer to generate a user interface to verify task rules with an application expert. After the rules had been verified the programmer wrote software to automatically capture design information from a design level, and the user interface became an optional means of viewing the task being performed.

The work described in this paper demonstrated that a task-oriented system can be automatically scheduled, configured, and programmed from the information generated by a design task level.

II. VIRTUAL TASK MACHINES

The task orientated approach suggests that rather than pursuing single robot structures which can 'do everything', the robot should be tailored to the permanent aspects of the task. Strickland presented three directives in his thesis which summarised the task orientated approach [2] and this was later extended to four directives in [5]:

Directive 1: Design industrial robots for the permanent aspects of the surrounding production environment and the task at hand.

Directive 2: The robot controller should be expandable to the requirements of the customer, from a set of generic modules and the robot itself should be constructed from modular robotic elements.

Directive 3: Robots should be task not functionally programmed using 'native' production languages or graphical interfaces. Such a rule-based hierarchy should interface directly to existing CAD/CAM facilities.

Directive 4: The machine should be able to accept prospective designs and advise on manufacturing details associated with the task. This advice may be, 'yes, I can manufacture the part, - it would take xxx seconds', or the machine could offer solutions to enable a production task to be performed, for example, new or alternative orientation information.

[6] Shows the development process for a task machine. The requirements, definitions, and specifications of the task are the starting point in creating a task machine. These specifications encompass the variance that would occur within the task being performed, that is they must not be product dependent. This information is then used in the design of both the hardware and the software for the machine. The design of the production machine hardware would take into consideration the available equipment, the hardware modules that have already been developed, and the fore-mentioned task specification. This would lead to the customisation of existing functional systems or the construction of a new machine using modular machine elements. If a suitable machine structure already exists then the controller can be removed and modular control boards used to control the machine. This second approach was used by Strickland to create a surrogate task machine.

A method which only caters for 'green field' (or new) factories only considers a small region of the potential market, and so methods of constraining existing general-purpose machinery have been developed. This involved the creation of virtual task machines. Virtual task machines use the existing machinery and controller, but constrain them for a task. A virtual task machine is not a true task machine in that the hardware is not built from modular

robotic elements. This means that the structure may not always be the most suited for the task, however, the interface to both types of machine is the same.

III. A NEW RESTRICTED PROGRAMMING LANGUAGE

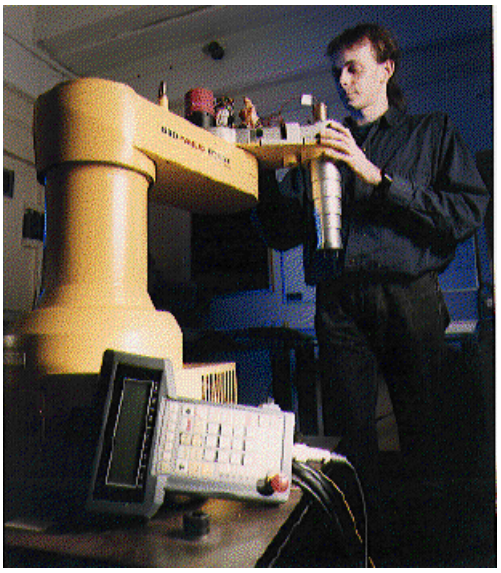
The new restricted programming language was a generically defined, broad base of commands for the control and simulation of advanced production machinery. General-purpose machinery was constrained by mapping the controller commands to the restricted programming language, using a predefined format and protocol. The modules not only contained the means to control the machinery, but also a model of the machinery, which allowed simulation. The modules were placed in a library, and were used in the structured programming environment to create virtual task machines. [4] Shows how the abilities of a robot controller were mapped to a broad base of functional commands.

A survey was conducted into robot programming methods and languages and is documented in [4,5]. The Fanuc programming language was representative of many programming languages available?

IV. A DE-FLASHING TASK MACHINE

A Fanuc A-600 scara configuration robot was selected for a de-flashing task. This had a different structure and kinematics compared to the Puma robots used in previous work. The Fanuc's native programming method was with NC type instructions, and the puma with a higher level programming language, Val I. The robot was selected because it contrasted with the Puma kinematics structures, and programming languages, whilst both had the ability to perform the task. A Fanuc robot is shown in figure 1.

Block diagrams of virtual task machines are included in [4]. Hardware interfaces were created to interface any RS232 controller to a transputer-programming environment. At the lowest level, software was required to drive the interface board, and to handle the communication protocols for the functional machine. This level effectively created a



'transparent' communication link with the machine, enabling commands to be issued from within the programming environment.

Fig. 1 Fanuc A-600 SCARA Robot

The functional commands were then 'captured' within the task programming system. This stage consisted of generating a translation table, which mapped the functional commands of the specific functional machinery to a general form for use within the task-programming environment. To do this, the structure of the language was analysed rather than the commands themselves. This structure was used to break down the languages and to translate them to a general structure. This translation table translated both commands and parameters. The front end to this translation table accepted commands as functional program

instructions.

At the highest level, task rules translated the task commands into functional commands. The task commands entered the system through a task interface. This interface was created to interface to a higher level within the system (for example a design level or work-cell co-ordinator level), or to interface with a programming environment driven by a user.

The design of the programming environment relied on the experience of an 'application expert' (someone well acquainted with the task being automated) to help shape the user interface. At this stage any obvious sub-tasks were also identified. The modular software framework was used to help configure the programming environment, and the task rules acquired along with any sub-tasks defined were also programmed. The application expert, away from the manufacturing equipment could then verify the rules. Higher and higher level sub-tasks were developed until the sub-tasks eventually collapsed into one main task.

V. CREATION OF MATERIALS HANDLING TASK MACHINES

A Kuikka conveyor system servicing four work-cells was used as a materials handler. A hardware interface controlled each work-cell station on the conveyor. A series of rules were developed to interface the functional control of each station to the work-cell co-ordinator. Software was written so that each station on the conveyor operated independently, as an intelligent station that could be interrogated to obtain the ability of the particular stations. The relationship between the materials handling task and the functional control was simple, and though the machines created were task machines the task was functional.

Various types of conveyor station were defined. The specifications were not specific to the hardware available for the research, but were developed as generic specifications that could be applied to any materials handling equipment. Three materials handling task machines were created, two standard stations, and one feeder. The task software was written in a modular, stand-alone form. This allowed the software to be used with a multiplexed single interface to all the actuators, or with separate interface hardware for each local station. A user interface was created to show the operation and the task programming of the stations. The real time operation of the interface was not possible because all the links on the driving transputer were being used.

VI. INTEGRATING THE TASK MACHINE AND A PRODUCTION ENVIRONMENT

The production environment created for the research consisted of a number of work-cells servicing task machines. The work-cells were linked to a factory co-ordinator. 0 shows the hierarchical structure of the environment.

The software was organised in the same structure, having work-cell co-ordinators, and a factory co-ordinator.

Work-cell Co-ordinators: The work-cell co-ordinators controlled the task machinery within a work-cell. They scheduled and sequenced events, handled errors, and serviced the requirements of the machinery (for example, service materials, such as cooling fluids). The work-cells created were not generic, configurable work-cells and did not have the full functionality which would be expected of a generic co-ordinator. The work-cells did provided identical functional behaviour to that of a generic

work-cell, for correct operation of the task machinery. The work-cells were able to schedule jobs and control the production using the available task machines.

Factory Scheduler / co-ordinator: The Factory co-ordinator, co-ordinated the tasks necessary for the manufacture of a product. A number of scheduling packages existed and it was not the purpose of this research to try and compete with these packages, but to show that the techniques used in these packages were naturally applicable to the experimental production environment created within this research. To demonstrate this, two functions were written into the factory co-ordinator. These were to schedule basic multiple tasks, and to show the automatic re-scheduling of tasks and parts within the manufacturing environment when machinery failed.

Two levels of priority were assigned to the tasks. The high priority jobs were assigned to the faster of two de-flashing task machines, and the low priority tasks to the slower. However the actual functional abilities of the two machines varied and so certain jobs (involving circular holes) needed to be routed to a certain machine. On the breakdown of one machine high priority jobs were rescheduled to another machine.

VII RESULTS (AUTOMATIC CONFIGURATION AND PROGRAMMING)

Initially each task machine was programmed and interrogated in isolation to the rest of the production environment. The de-flashing task machines were given task data, which would have originated from a design level. 0 shows an example of the data for a tool to produce a simple plastic box. (a) and (b) show the two tooling sections generated at a design level, and (c) shows the contact data where the two sections of the tooling touch. It was this information which was used by the task machines. Information concerning the location of aesthetically important faces was also sent, along with data, which defined the edges of the tool.
Various design data were given, some of manufacturable designs, some of designed where the aesthetically important faces would be damaged, and some of designed impossible to manufacture. When asked to advise on the design, the task machines either responded with a confirmation that the part could be manufactured, and the time it would take to do so. Warnings were indicated to identify aesthetic faces that would be damaged.

When asked to advise on a design, the task machine responded with information concerning the manufacture of the part. This information in its simplest form would simply acknowledge that the part could be de-flashed and report the time it would take to complete the operation. If only aesthetic damage was detected then the same information was returned, along with warnings that the laser cutting process would scorch particular surfaces. If however manufacture was not possible then an error was returned. Advice was provided on reorientation to eliminate manufacturing and aesthetic errors.

To demonstrate that the machinery was being programmed in terms of task and that the programming was independent from the kinematics structure of the robot an experiment was created so that the same data could be sent to either task machine. Both task machines responded in the correct way.

The de-flashing and the materials handling task machines were integrated into the production environment. The environment was given a number of low and high priority jobs to perform, and it successfully scheduled and completed these tasks. The work-cells and factory co-ordinator were able to advise on the

time it would take for production, and the factory co-ordinator automatically scheduled parts within the environment.

The system was able to automatically reconfigure itself, and was fault tolerant. During production one de-flashing machine was removed to simulate machinery failure. The system automatically rescheduled the high priority jobs to another task machine, de-scheduling the low priority jobs. When the machine was replaced, the system again re-scheduled work back to that machine. A video demonstrating these results is available upon request from the author.

VIII. DISCUSSION AND CONCLUSIONS.

A method for programming automated machinery has been presented. This was successfully applied to a small manufacturing environment. The virtual task machines created were programmed in terms of the task to be performed rather than their functional operation.

The initial results proved that the new programming techniques described in this paper can be used at a factory level as well as for a single task machine. The successful automatic scheduling and programming of the machinery with data supplied directly from a design level demonstrated that this is a potential route towards full factory automation.

The production environment created as part of this research supports further research being conducted at the University of Portsmouth into the integration of the design level. Methods of integrating the advice from task machines and the factory co-ordination elements of a factory, with an interactive design level are being investigated, along with new methods for capturing the knowledge of a designer.

REFERENCES

[1] P Strickland and J E L Hollis, "*Task Oriented Robotics*", Proceedings of the SICICI conference, Singapore, Vol 2, pp 835-840. 1992.

[2] P Strickland P, "*Task Orientated Robotics"*, Portsmouth Polytechnic, PhD, UK. 1992.

[3] S A Bonner, "*Comparative Study of Robot Languages",* Rensselaer Polytechnic Institute, Troy. S A, MS Thesis, 1983.

[4] M P Deisenroth, "*A Survey of Robot Programming Languages*", Proceedings of The 1985 Annual International Industrial Engineering Conference, IIE, pp 191-194. 1985.

[5] G E Tewkesbury GE, "*Design using distributed intelligence within advanced production machinery*", PhD, University of Portsmouth, UK, 1994.

[6] G E Tewkesbury, P Strickland, D A Sanders and J E L Hollis), "*Product Orientated Manufacturing*", Proceedings of the 25th Dedicated Conference on Mechatronics (part of ISATA 92), Florence, Italy, pp 505-512. 1992.

[7] G E Tewkesbury , D A Sanders, P Strickland and J E L Hollis, "*Task Orientated Programming of Advanced Production Machinery*", Proceedings of the 26th Dedicated Conference on Mechatronics (part of ISATA 93) Aachen, Germany, pp 623-630. 1993.