Robotics and Computer-Integrated Manufacturing 25 (2009) 560–569

Contents lists available at ScienceDirect

Robotics and Computer-Integrated Manufacturing

journal homepage: www.elsevier.com/locate/rcim

A novel open CNC architecture based on STEP-NC data model and IEC 61499 function blocks

M. Minhat [a], V. Vyatkin [b,*], X. Xu [a], S. Wong [b], Z. Al-Bayaa [b]

[a] Department of Mechanical Engineering, The University of Auckland, Private Bag 92019, Auckland Mail Centre, Auckland 1142, New Zealand
[b] Department of Electrical and Computer Engineering, The University of Auckland, Private Bag 92019, Auckland Mail Centre, Auckland 1142, New Zealand

ARTICLE INFO

Article history:
Received 29 January 2008
Received in revised form
11 March 2008
Accepted 26 March 2008

Keywords:
IEC 61499 standard
Function blocks
STEP-NC
CNC machine
Distributed control systems

ABSTRACT

Modern manufacturing industries demand computer numeric controllers, having higher level input languages than outdated G-code, and less proprietary vendor dependencies. IEC 61499 is a new standard for distributed measurement and control systems, that enables portability and interoperability of embedded controllers, along with the ease of their mapping to arbitrary distributed networking hardware configurations. This paper demonstrates that the IEC 61499 reference architecture can be successfully used to create a computer numeric controller, offering interoperability, portability, configurability, and distribution characteristics. The layered CNC-FB architecture is proposed, which simplifies the design of a CNC machine controller with the architecture layers responsible for data processing, data storage and execution. In combination with the object-oriented Model-View-Control design pattern, the CNC-FB architecture supports the design framework, in which simulation of the machining becomes natural and inherent part of the design process, with seamless transition from simulation to actual machining. The implemented controller was tested in both the model and on an actual milling machine.

© 2008 Elsevier Ltd. All rights reserved.

1. Introduction

Computer numerical control (CNC) devices are the brains of machine tools used in all manufacturing industries. Since the 1950s, CNCs have gone through several generations, following the state-of-the-art computational platforms. The way of their programming, however, remains almost unchanged. The low level G-code programming language is still used in most of the CNCs. The G-code language was designed in the era when paper tape was the medium for moving data between computers and CNC systems. Today an average microprocessor can easily process complex three-dimensional data, and CNC machining is the only operation in the design-to-manufacturing pipeline that is not using full-fidelity product and process information.

A typical design-to-product chain is shown in Fig. 1. Nowadays the G-code programs are generated by the computer aided manufacturing (CAM) tools, using the geometrical data from computer aided design (CAD) tools as the input. However, CNCs of different vendors implement different versions of G-code which lacks any portability and leads to proprietary CAD–CAM–CNC chains [1,2]. In order to generate G-code programs for different CNCs, CAM tools need to know not only the particular brand and model of a CNC, but also need to have a detailed description of the machine tool and their peripherals such as cutting tools and other auxiliary components. This information is being handled by a special unit within a CAM tool called "postprocessor" which operates with full knowledge of: (i) G-code, (ii) machine-tools and (iii) libraries of cutting tools. The postprocessor generates exact scenario of machining which is described in a G-code file.

As illustrated in Fig. 1, software tools (such as CAD, CAM), CNCs, and machine tools, are designed with versatility in mind. For example, CAM tools can read output data formats of different CAD tools, machine tools can be equipped with, and driven by different CNCs. Machine tools can also use other devices provided by different vendors. CNCs, however, can accept programs only in their proprietary versions of G-code. This hinders the flexibility of manufacturing processes, especially within extended global enterprises. In order to manufacture a certain product on a certain manufacturing site, the company may need to pass on their full information, database and data model of the product. The G-code language is clearly insufficient for this. Even though existing CNC systems have become more and more sophisticated, incompatibility between their proprietary data restricts further productivity enhancement of CNC-based machining. Furthermore, since each system has its own data format, the same information must be entered multiple times into multiple systems leading to redundancy and possible errors.

* Corresponding author. Tel.: +64 9 3737599.
E-mail address: v.vyatkin@auckland.ac.nz (V. Vyatkin).

0736-5845/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.rcim.2008.03.021

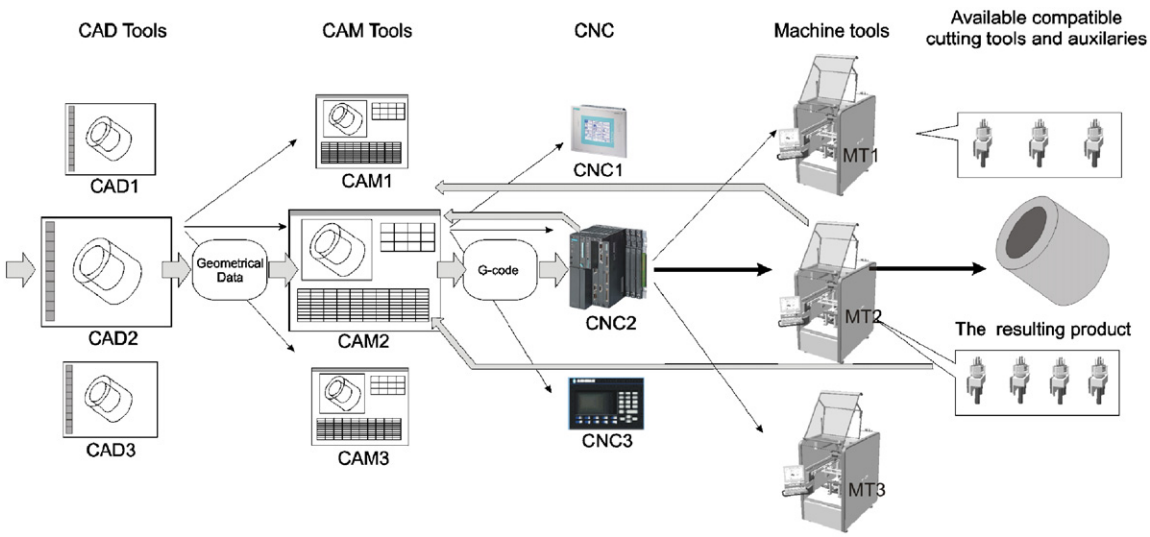

Fig. 1. Automated design-to-product chain.

The implementation chain would have been much easier if a standard format for describing geometrical and machining information had existed, and if the CNCs could have directly read and understood such documents and immediately carry out machining. For this reason, a new language, STEP-NC, has been created and standardized as ISO 14649-1 [3]. STEP-NC is envisaged to be a single language used throughout the design and manufacturing implementation chain. However, in order for it to be used on modern CNCs, specific software tools are required to translate STEP-NC descriptions to the proprietary G-code of the particular CNC.

Today's CNC manufacturers offer most of the features either in a built-in package or in the made-to-order manufacturing mode. Hierarchically, CNC machine tools are facing numerous challenges. They become more and more complicated and require decentralized control [4,5]. Their internal structure needs to be flexible and customisable. The control needs to be implemented as a network of several embedded computing devices, interchangeable depending on different applications area or target markets. Machine tools are often a part of an automated manufacturing system and therefore have to interoperate with other automated machines, such as robots, transport, storage and so forth. As such, they can be substituted by other similar tools from other vendors. Thus CNCs have to be easily integrated into automation systems, interoperating with programmable logic controllers (PLCs), other CNCs, embedded control devices, etc.

In this paper we propose a novel open architecture of CNC which aims to meet the above mentioned challenges and provides some insight into a new generation of CNC with open, flexible architecture. The idea is illustrated in Fig. 2. The new generation CNC will be capable of accepting STEP-NC data directly. The software architecture of this CNC is based on the open reference architecture known as the IEC 61499 "function blocks" standard [6]. The IEC 61499 function blocks will enable easier program distribution, integrated visualization and improved real-time control characteristics. CNC milling machines or machining centres are the targeted machine tools in this research. The hardware architecture of the future CNC will be flexible and distributed. It allows for scaling its computing power, memory and interfaces up to particular needs.

2. Related works on CNC systems

The primary challenge for CNC systems is the agility in implementing various manufacturing strategies driven by market changes. New research results [7,8] show that the agility can be achieved if the machine controller unit is based on open architectural concepts [9] rather than proprietary hardware or software architectures.

Since G-code programs describe only plain machining commands, a richer programming language is required to cater for other automated operations. Such a language needs to be open and rich to handle CNC machines of different vendors. Therefore, vendor-neutrality and component-interoperability can be thought of as two fundamental features of an open system [10]. Ultimately, the main goal for research projects on open CNCs is to develop a vendor-neutral, tool-neutral and controller-neutral architecture of CNC.

A common CNC executes G-code programs, which are generated as a result of planning and scheduling in CAM tools. When designing a CNC machine, it is important to consider its dynamics, electrical components and mechanical structure [11].

The idea of an open CNC [12,13] is to create an integration platform, on which the specific software components (handling the particulars of specific machines) can be easily integrated with a core CNC functionality (planning, scheduling and execution). An open CNC will be able to interoperate with extensible set of machine tools, cutting tools and CAD formats. One possible approach to the integration is to use a personal computer (PC) as a hardware platform with an operating system simplifying the integration.

Thus, Gordon and Hillery [14] introduced Microsoft Windows-based CNC, programmed with the MINT language, which is a structured form of BASIC designed for motion control applications. The motion control was implemented using the Baldor Nextmove BX 3-axis servo motion controller that was supplied with a linear motor system. The motion control unit communicated with a PC via an RS-232 serial link, and was capable of storing and executing entire part programs. Since many CNC functions are very computation intensive and require hard-real time performance, the real-time operating system QNX has been used in some research works [7]. Powerful multitasking mechanisms of PC operating systems (such as Windows or QNX) bring extra advantages to CNC implementation, e.g. such a CNC can provide disturbance estimation and monitor subsystem to control and monitor features such as torque and over-cutting state. The proposed kernel software [10] organizes and manages various control software modules dynamically by using process and resource models. The kernel software enables the CNC to be easily

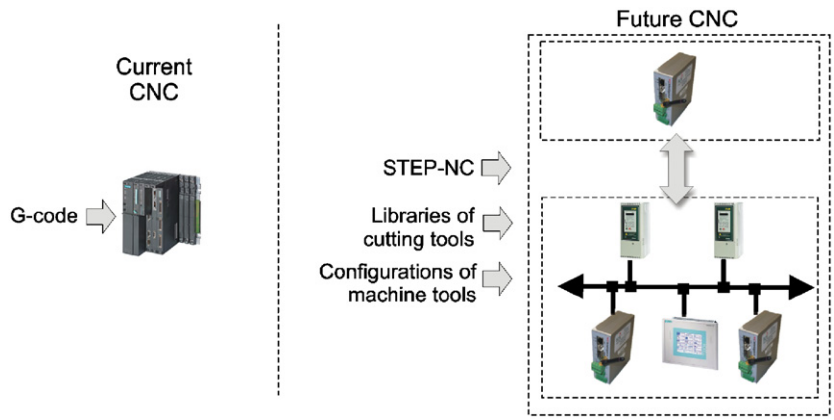

Fig. 2. Current CNC vs. future open extensible CNC.

reconfigurable and adaptable to new machines by adding new interface software modules.

Making CNCs more intelligent is another quest from both CNC makers and users. Some of the recent developments include a digital signal processor (DSP) for an EDM (electro-discharge machining) machine [15], Virtual CNC [16] and an intelligent STEP-NC controller [9].

Chang proposed in [15] a monitor and controller of the ignition delay percentage of the discharge pulses, to maintain the rate of erosion in the EDM process. A flexible program calculates an accurate ignition delay percentage and then presents the results on a digital-to-analogue converter. Erkorkmaz and Wong presented in [16] a technique for rapid identification of machine tool drives. The proposed strategy uses commanded and measured axis profiles and requires minimal intervention to the servo control loop. The methodology is fairly general and applicable to linear or ball screw drives, controlled with commonly used controllers such as P, PI, PID, P-PI Cascade or Adaptive Sliding Mode Control; with or without feed forward dynamic or friction compensation. The identified models were used in a Virtual CNC system for predicting the contouring and tracking errors to different part programs.

There is a great deal of research on the CNC architectures based on STEP-NC. Han et al. in [9] proposed a novel framework based on multi-agent for implementing an intelligent STEP-NC controller. The architecture allows for analysis of process routine and the specific requirements for the intelligent STEP-NC controller.

Most of the above-mentioned CNC systems are still under influence of G-code. The ultimate goal, however, is to machine directly from the part program bypassing 'postprocessor' and G-code. Currently there is no CNC which can process CAD-enriched machining data directly. In this paper, we present an open CNC architecture that is based on STEP-NC data model and IEC 61499 function block reference architecture. The architecture: (i) supports bidirectional information flow in the design and manufacturing chain; (ii) adopts the concept of feature-based machining for CNCs so that higher-level information can be made available at the CNC machines; (iii) enables an autonomous and more intelligent CNC; (iv) supports a distributed process planning scenario; (v) is modular, reusable and open; (vi) is scalable, extensible and portable.

3. Overview of the IEC 61499 architecture and MVC design pattern

3.1. IEC 61499 architecture

A function block is a software unit that encapsulates algorithms which can be designed to behave in a similar way as an electronic device or a circuit. This means that a function block can represent a small task in a control plan or it can encapsulate multiple small control units. The unit, designed for a specific purpose, contains the process algorithm and control state machines needed to accomplish a specific task (Fig. 3).

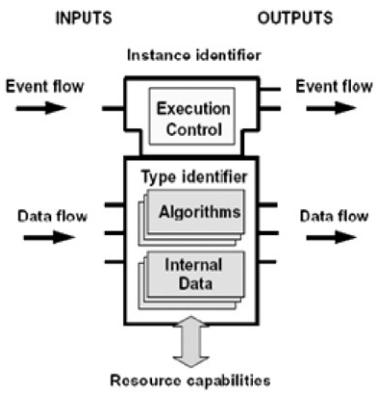

Fig. 3. Function block model.

There are three standard classes of function blocks defined in IEC 61499: basic function blocks, composite function blocks and service interface function blocks. Each function block has a set of input and output variables. The input variables are read by the internal algorithm when it is executed, while the results from the algorithm are written to the outputs.

The basic block type encapsulates algorithms and has an execution control chart, giving this block type the flexibility to model many different components. An example, defining basic function block type $X2Y2$, is seen in Fig. 4. In basic function blocks of IEC61499 a state machine (called Execution Control Chart, ECC for short) defines the reaction of the block on input events. The reaction can consist of execution of *algorithms* computing some values as functions of input and internal variables, resulting in emitting of one or several output events. In Fig. 4 (right side) the ECC and an algorithm are shown. The state REQ has one associated *action* that consists of calling the algorithm REQ followed by emitting of the output event CNF afterwards. The algorithm computes $\text{Out} = X^2 - Y^2$.

A composite function block can encapsulate a network of multiple blocks (both basic and composite), interconnected by external data sources. An example is given in Fig. 5, where the same $X^2 - Y^2$ function is implemented as a network of three function blocks, doing addition, subtraction and multiplication, respectively. This network can be encapsulated in a composite function block with the same interface as the function block $X2Y2$ from Fig. 4. The possibility

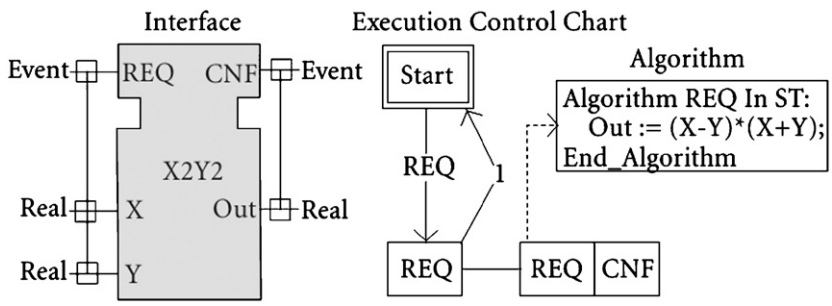

Fig. 4. A basic function block type description: interface, ECC and algorithm REQ.

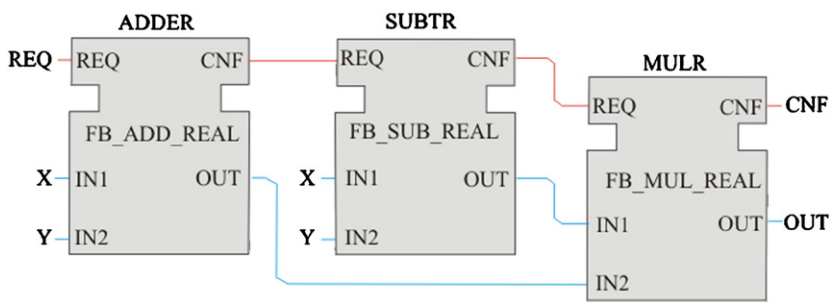

Fig. 5. Implementing X^2–Y^2 as a network of function blocks.

to include composite FBs within other composite FBs enables a hierarchical system description. This is useful for defining multi-layered architectures. In our case, the developed CNC architecture is layered with layers responsible for data processing, data storage and execution. The separation of the functions into layers enables flexibility of the controller. The FB-based architecture also enables modelling and simulation to be tightly integrated with the design process. Before execution, the controller can be validated by either simulation or formal verification.

In the IEC 61499 architecture, the function performed by the system is specified as an application, which may reside in a single device or be distributed among several devices. The application consists of a network of function blocks connected by data and event connections. The control system is specified as a collection of devices interconnected and communicating with each other by means of one or more communication networks.

The use of function blocks makes the control device openly programmable and easily reconfigurable. IEC 61499-compliant devices can easily interface one another, thus providing for seamless distribution of different tasks across different devices [17,18]. The user may create own program using the standard function blocks. Thus, the IEC61499 architecture enables encapsulation, portability, interoperability and configurability. Portability means that software tools and hardware devices can accept and correctly interpret software components and system configurations produced by other software tools. With interoperability, hardware devices can operate together to perform the cooperative functions specified by one or more distributed applications. With configurability, devices and their software components can be dynamically configured (selected, assigned locations, interconnected and parameterized) by multiple software tools.

3.2. Software implementation: FBDK and FBRT

Function block development kit (FBDK) [19] is the software tool, widely used in the IEC 61499-related research projects. The tool allows for the graphical development of both function blocks and function block-based systems. FBDK compiles the developed function block types into Java programming language. The object-oriented nature of this language enables consistent implementation of function blocks. In addition, the platform independence of Java leads to portability of the implemented controller.

The use of FBDK can be combined with other Java development tools, for example, Eclipse Integrated Development Environment, especially when sophisticated Java code (e.g. 3D visualization) or interfacing peripheral devices (e.g. parallel port) need to be encapsulated into function blocks. The function block applications, designed using FBDK, are executed with Java virtual machine and the function block runtime (FBRT) library of standard FB types.

3.3. Model-View-Control design pattern

The object-oriented Model-View-Control (MVC) software design pattern was adapted by Christensen in [20] for industrial control. Its applicability has been proven for the IEC 61499 architecture. More details on implementing MVC with function blocks can be found in the book by Vyatkin [21]. The pattern is represented graphically in Fig. 6. The core part of the pattern is the closed-loop object–controller interconnection. In software, the object is represented by an *interface* to its data sources (say sensors) and signal consumers (actuators). The object can be substituted by its model—a software entity having the same interface and simulating object's behaviour. Several models can be used depending on the required accuracy and the purpose of modelling.

The proposed novel CNC framework in this research was developed using the concept of MVC, which enables full simulation and rendering of the CNC system, driven by the actual controller code.

4. Layered architecture of the novel CNC

The architecture of the proposed CNC controller (i.e. the Control part in the MVC architecture) is layered and is implemented

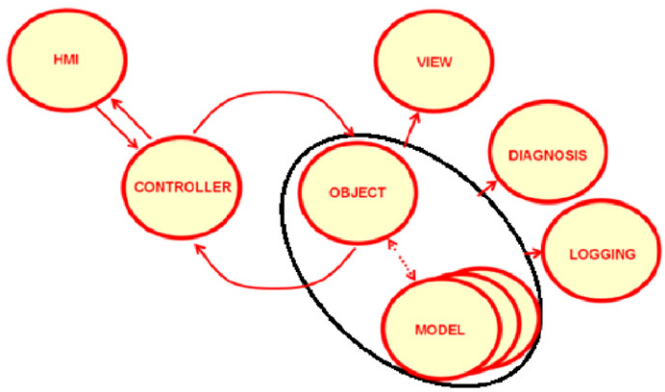

Fig. 6. Architecture based on the MVC design pattern.

using function blocks. The use of the layered software architecture as the framework for CNC design enables flexibility in adopting the controller to new machines and computer hardware/software platforms. The layers are selected to separate the functional units into a hierarchical structure so that the controller becomes more readily maintainable, as modifications can be done to a particular layer without compromising the functionality of other layers. Each layer is designed to utilize the services of the lower one.

There are three groups of functions as the backbone of the layers: (i) input and data distribution of machining features, (ii) data storage/buffering and (iii) output of physical signals and process execution. The complete architecture consists of five layers as shown in Fig. 7. In addition to the three groups of functions mentioned above, two more layers are added: process planning (Layer 5) and 3D co-ordinates (Layer 3). Functions of all layers are encapsulated in function blocks. Communications between all of the layers are implemented by the communication function blocks *Publish* and *Subscribe*.

The layers are further divided into two categories as shown in the right-hand side of the diagram: "Input Model and FB Generic Data Program" and "Machine Specific or Native Program". The former contains primarily generic data whereas the latter contains machine specific data. All layers (5–1) are described as follows.

Layer 5 (planning) is to provide definitions on security plane, cutting tool, workplan, workpiece, clamping position and so forth. Layer 4 (Machining features) is implemented by a library of function blocks that parameterize machining features and machining data. It generates the tool paths required for the respective features. Function blocks are used for directing and managing features and shape coordinates to accept or collect data entered either manually by user, or automatically from other data sources.

The input types required for initialization purposes include:

- Travel limits (the maximum travel displacement of each axis).
- Work piece dimensions.
- Work piece origin.
- Tool start point (home positions).

An additional set of input options is used to enable the operator to assemble the tool path of the shape. It consists of the following options:

- *Customized shapes blocks*: Each block represents a basic shape, such as a rectangle or a circle. Each of those blocks generates the sequence of coordinates, required to construct the shape using the dimensions, provided as an input.
- *Text file*: The library of this layer also contains a file reader block designed to read the coordinates of a design part from a text file.
- *Point-to-point*: Manual point-to-point mapping required to build the tool path.

Layer 3 (store/buffer) stores all the points and feed rates required to execute a complete job on the machine tool as a coordinator, with function blocks making and receiving requests from the point memory storage. The relative velocities are then passed to the Layer 2.

Layer 2 (process execution) is a coordination layer. Function blocks make and receive requests from the point memory storage, and calculate the feed rate for each individual axis, before sending updated physical signals to the output device. In addition, it waits for a confirmation from Layer 1 (physical output) before proceeding to the next point.

Layer 1 (physical output) activates the motion control. In the case of the milling machine the required frequency and number of pulses are calculated from the given distance and feed rate.

5. The prototype system

A prototype of the proposed CNC architecture has been implemented on a testbed, a PC controlled CNC vertical milling machine. The CNC machine is a 3-axis mill. The machine is supplied with a PC-based EMC (enhanced machine control) controller and a motor control unit that drives three stepper motors. The hardware of the PC and the motor control unit are used for the prototype system. A PC is used to run a Java-based function block execution platform. It is envisaged that embedded devices, capable of running function blocks, will be capable of deploying the same solution.

5.1. Software implementation

The functional layers of the proposed CNC are encapsulated in function blocks as discussed in the previous section. Based on the design data of a part, the corresponding function block application is generated. At this stage, such applications were created manually, following the developed architectural pattern. It is envisaged that in the full implementation, this step will be performed by software tools, translating STEP-NC data into the function block application. The function block application uses instances of function blocks defined in Layer 4 in the proposed CNC architecture. The numeric data of the STEP-NC file are translated into parameters of the function blocks.

The features, implemented by function blocks in Layer 4, are translated into trajectories, i.e. sequences of points and stored in Layer 3. Moving between the points is implemented by the motion controller function blocks Layer 2, which will calculate an output to the axes motors from the sampled positions, generated by the calculated trajectory points. The output, generated from the motion controller is passed onto the discrete input/output (I/O) interface function blocks in Layer 1, which send the control signals to the motor's drivers.

The proposed CNC structure is presented in Fig. 8. The CNC is implemented as a system configuration of four devices (in IEC 61499 terminology): Model, Display3D, Controller and Physical-Out, corresponding to View, Model, Control and Interface as in the extended MVC architecture. The process starts with STEP-NC data model of the part that is represented (manually for now) by the Layer 5 function blocks. It is then translated to the function block application Control (allocated in the "Controller" device), which is

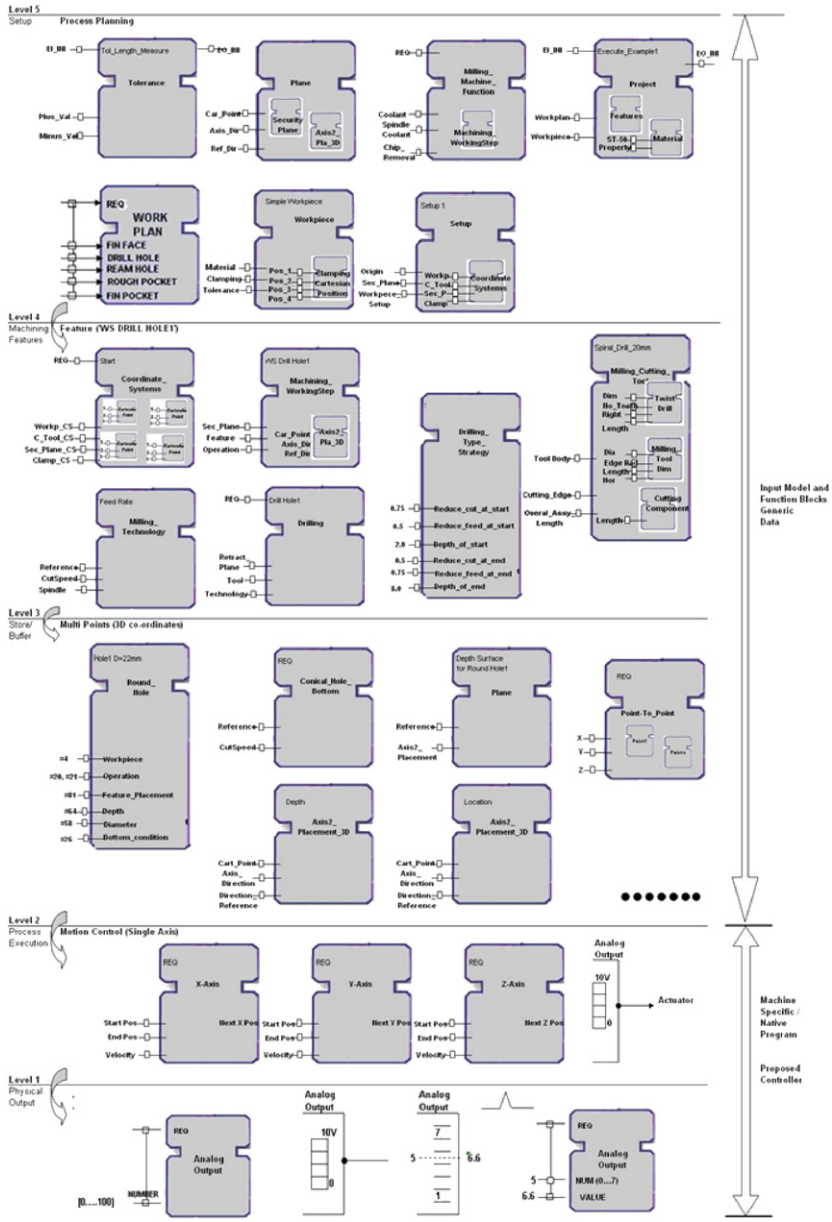

Fig. 7. Layered CNC-FB architecture of the novel CNC architecture.

composed of the function blocks from Layer 4 and is using services of Layer 3. Execution containers (resources in IEC 61499 terminology) are used to separate these layers out into distinct tasks. The Layer 5, for example consists of the following resources:

- Machine (travel dimensions).
- Workpiece (dimensions).
- Workpiece_start.
- Tool_start.

The "Model" device contains the function block application, modelling uncontrolled behaviour of the machine tools. Given input control signals the model changes its state as the real machine tool does. It feeds the parameters of the current state to the device DISPLAY3D which can render FB applications.

The PhysicalOut device sends the control signals, generated by the function of Layer 1 in the controller to the parallel port of a PC, on which the configuration is executed. The devices communicate with one another by means of communication function blocks.

This system configuration, with minimal changes, can be executed on any computing platform supporting the IEC 61499 function blocks. As shown in Fig. 8, it can be a PC with direct peripheral connection to a machine tool (right side), or an embedded device with a distributed architecture (left side). In the latter case, the motor drives, corresponding to three axes and the display are connected to the main processing unit via a field area network (fieldbus). With this motor control, only a direction signal and a single pulse are required to move a motor a single step in any direction. The desired behaviour of physical interface is to utilize the raw data calculated in the controller (feed rate (mm/min)) to control the real machine (i.e. to produce the signals required to drive the motors). Two parameters are required to be supplied to the input interface of the motor driver circuit: direction and speed. Direction is determined by a signal being true or false, while speed is represented by a sequence of pulses.

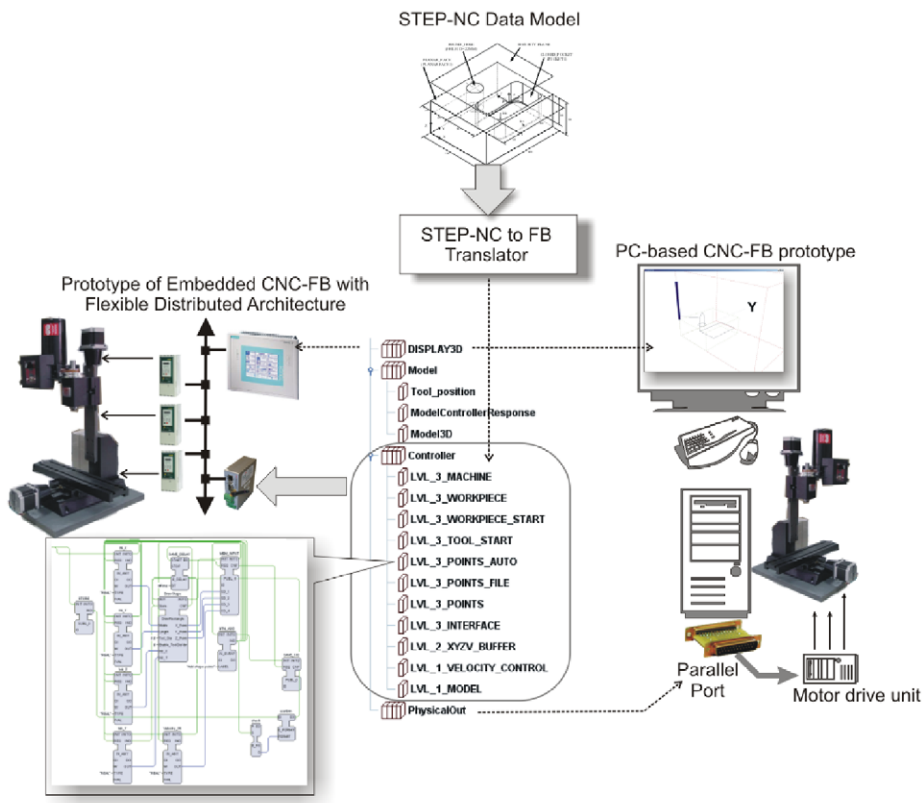

Fig. 8. System structure based on the MVC pattern.

In the prototype system, the physical interface of the PC with the motor drives is implemented using its parallel port. The dependency on this particular hardware solution was captured by creating an abstract device model, called *ParallelPortDev*. The implementation of the device relies on the RXTX library [22], providing programming access to the hardware. The device model allows a specific library of function blocks to provide user interfaces to the motor drives via the parallel port of the PC. The access process includes opening the parallel port stream and calculating the required direction and number of steps to traverse. This is communicated to the *ParallelMotor* function block (Fig. 9) which generates the required output sequence of data to execute the motor control, and in turn drives the motors on the machine. At the completion of the sequence the parallel port stream is closed. Other function blocks in the library were implemented to encapsulate direct communication to any or all output pins of the parallel port.

A composite function block, using the services of the ParallelMotor function block, is designed to handle velocities and number of steps of all three motors. It can be easily extended to accommodate the fourth motor. This function block generates an output sequence which is written to the parallel port, and in turn drives the motors on the machine.

5.2. Implementation of the MVC design pattern

The Model of the machine is the core part of the MVC architecture. Machine modelling is done by taking into account machine's structure, its physical dimensions, number of axes available and the type of inputs, outputs and movement expected. Geometrical parameters of the features are used as input parameters in the corresponding function blocks. The workpiece data model will provide point-to-point connection to simulate the tool path movement, same to the role of a Tool Path Generator in a traditional CNC. The controller needs to generate two types of signals: target signals and velocity signals (one for each individual axis). Target signals correspond to the coordinates (X, Y, Z) of a destination, and the velocity signals represent the vectored feed rate of each axis.

The *Movement_Estimator* function block (Fig. 10(a)) determines the data for the planned path, which is then fed into an axis model block that estimates the next point to be represented on the axis. The *Axis_Model* (Fig. 10(b)) block is used to represent a linear axis control unit to define an axis in the system by capturing its details. Therefore, in order to create the 3D representation of a model, three *Axis_Model* blocks are needed to denote the (X, Y, Z) coordinates of the next point. This block will take the speed and direction inputs for an axis and update its coordinate position. It can then be used for each of the X, Y and Z axis control.

Further limits were added to the model to reflect the constraints of the machining boundaries. These include the maximum travelling speed and the maximum travel distances along each of the three axes.

5.3. Visualization

The Visualization component of the MVC architecture is implemented in a 3D virtual environment through the Java extension, Java3D. The developed visualization engine can render workpieces and cutting tools. Travel constraints and axes relationships are also modelled.

The engine is accessible through the ImageDev3D device, and utilizes a library of rendering function block types such as *RenderAxis3D*, *RenderWireBox3D*, *RenderCylinder3D* and *RenderLine3D*.

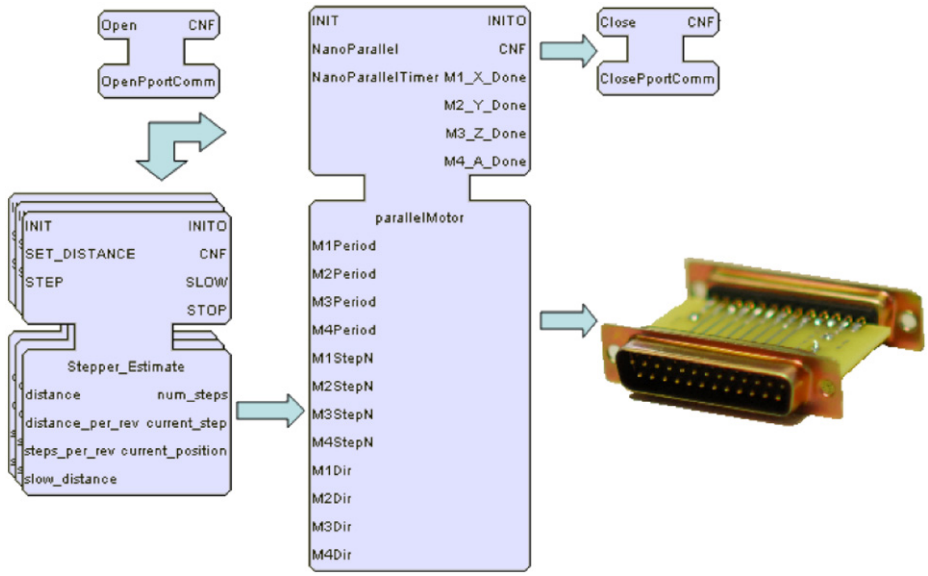

Fig. 9. ParallelMotor function block.

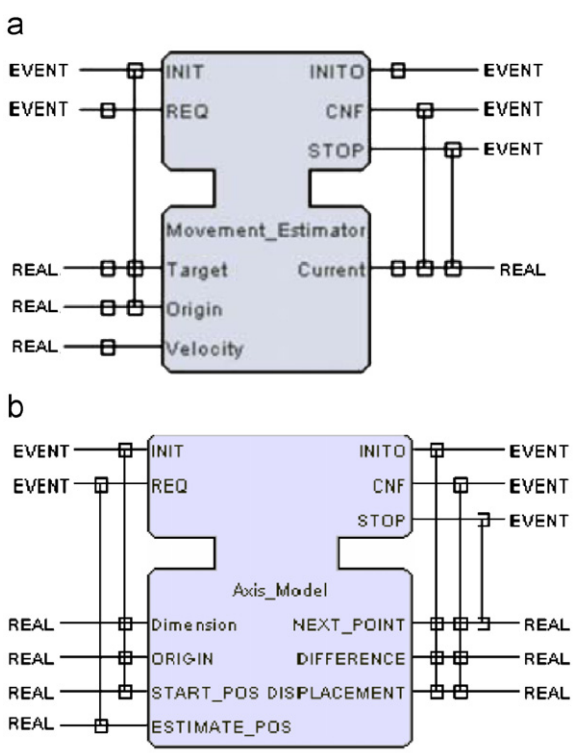

Fig. 10. (a) Movement_Estimator function block and (b) Axis_Model function block.

3D rendering is achieved in a three-step process. First a JFrame object is extended to support 3D rendering. This is done by creating a 3D Universe within it. JFrame itself is a native object in the 2D library Java.Swing, and handles the creation of a window object in which to render. Next, an interface to the function block environment is created. This interface is called the ImageDev3D. This is encapsulated as a function block device (as it represents a separate subsystem). An extended JFrame object is added as a child of the ImageDev3D. In the final step, separate function blocks are created that define a particular shape. These function blocks construct Shape 3D objects which can be rendered in

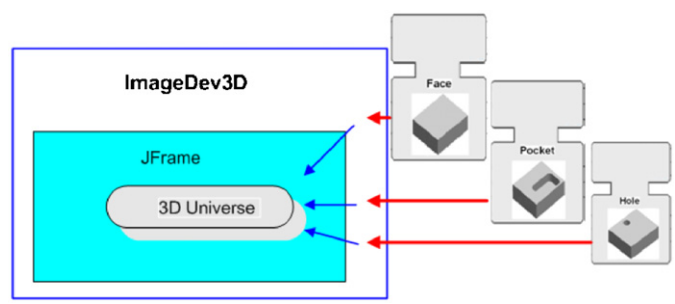

Fig. 11. Overview of function block 3D rendering.

the 3D Universe. The geometric data are transferred to the JFrame that exists in the ImageDev3D. The entire process is depicted in Fig. 11. In this way it is possible to render multiple instances of any shape, as long as there is a function block for each instance. Furthermore it allows other more complex geometries to be created with ease.

Each shape added to the 3D Universe is coupled with a transform object that allows a user to manipulate the translation, rotation and scale of the shape. Modifying the transform object allows dynamic motion of a shape without comprising the original shape data. In addition the 3D visualization supports some typical navigation abilities as seen in many other 3D packages—namely rotate, zoom and pan. This enhances the user-friendliness of the interface.

5.4. Sample part

The starting point of the process is STEP-NC data, which contains information such as "Workplan", "Workingstep", machining strategy, machining features and cutting tools. The prototype system was tested by manufacturing several parts, one of which is the first example in the annex of ISO 14649-11 [23] as shown in Fig. 12(a). The data shown in the figure is included in the STEP file.

The current prototype system only uses wireframe models to represent a workpiece and the maximum travel constraints (Fig. 13). The cutting tool is represented by a cylinder. Two types of

lines are visible in the model, one representing the ideal tool path (Fig. 13(a)), and the other the actual traversed path (Fig. 13(b)). Motion is visualized by constantly updating the transformations for each of these elements. The 3D visualization mirrors what one would expect of the milling machine. This allows for a preview of the controller motion, and can be used as a virtual verification method.

Testing of the system was first completed using the 3D visualization. Following this the Sherline CNC vertical mill was engaged with the same part information as used in the simulation. The resulting part matched the simulation in terms of dimensions of the shape machined as well as machining speed.

6. Conclusions and future work

To meet the challenges of the ever globalized manufacturing economy, agile, distributed, interoperable and intelligent computer numerical controlled machine tools are one of the key enablers. Possible ways of replacing the current control language, i.e. G-codes, has been researched in recent years. The proposed CNC controller uses the IEC 61499 architecture as its development platform and STEP-NC as the input data model. Compliance with the STEP-NC standard will help to transfer the data across different companies and manufacture the corresponding part on different machine tools. The prototype system has a layered structure. This type of organization simplifies the transition between simulation and real machining, and substitution of one milling machine by another.

The CNC architecture proposed in this paper has been tested through a prototype system, using a CNC vertical milling machine and proved that use of function block technology can allow development of an open and distributable CNC system. This also enables separate functional units of the controller to be implemented on other devices.

In addition to the distributable nature of the prototype system, the proposed controller can also support remote management and configuration, a feature that could be useful in an environment featuring multiple machine tools (e.g. flexible manufacturing systems). Thanks to the successfully implemented MVC design methodology, the architecture and its prototype system support real-time machining simulation. Hence the system can also be used as a virtual machining tool.

To cater for tool path planning, a library representing basic machining features should be developed as well as a higher level machining planning. Further work will also include deployment on embedded devices with pulse width modulation (PWM) generators to enhance the performance of the prototype system.

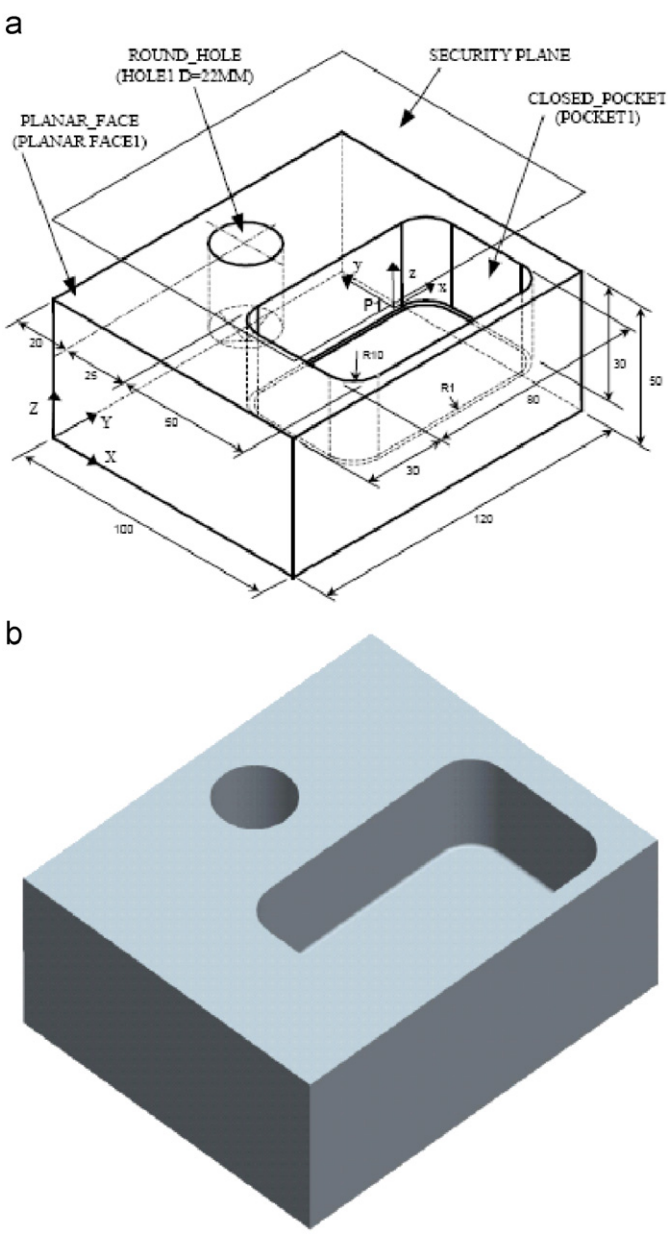

Fig. 12. Sample part from ISO 14649-11.

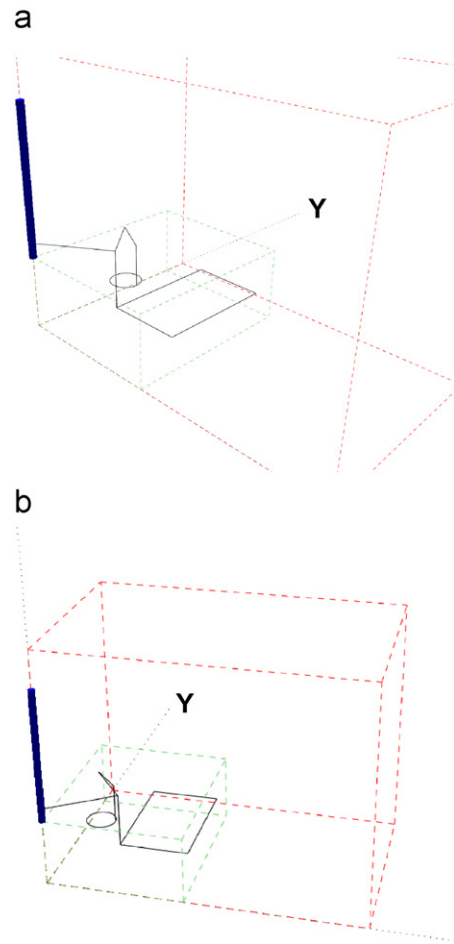

Fig. 13. 3D visualization of the milling process.

Extensibility will be addressed by extending 3-axis to 4 and/or 5-axis control function, which will be achieved by implementing the corresponding function blocks at Layer 3 of the architecture. With respect to interoperability, the output interfaces to other machine tools need to be implemented as function blocks and device types, belonging to the layer 1 of the CNC-FB architecture.

References

[1] Wang H, Xu X, Tedford JD. An adaptable CNC system based on STEP-NC and function blocks. Int J Prod Res 2007;45(17):3809–29.
[2] Xu XW, Wang L, Rong Y. STEP-NC and function blocks for interoperable manufacturing. IEEE Trans Automat Sci Eng 2006;3(3):297–307.
[3] ISO 14649-1. Data model for computerized numerical controllers: part 1—overview and fundamental principles. International Standards Organization; 2003.
[4] Kovacic M, Balic J. Evolutionary programming of a CNC cutting machine. Int J Adv Manuf Technol 2003;22(1–2):118–24.
[5] Wang L. Integrated design-to-control approach for holonic manufacturing systems. Robotics Comput Integrated Manuf 2001;17(1–2):159–67.
[6] IEC 61499. Function blocks for industrial-process measurement and control systems-part 1: architecture. Geneva: International Electrotechnical Commission; 2005.
[7] Chen F, Yang Y, Weng P, Fu P. Application for control system analysis using system analysis toolkit. J Comput Inf Systems 2007;3(1):49–56.
[8] Kramer TR, Proctor F, Xu X, Michaloski JL. Run-time interpretation of STEP-NC: Implementation and performance. Int J Comput Integrated Manuf 2006;19(6):495–507.
[9] Han S, Liu Z, Yang X, Zhao J. Research on architecture of open CNC system based on component technology. Nongye Jixie Xuebao/Trans Chin Soc Agric Mach 2007;38(10):127–31.
[10] Park S, Kim S-H, Cho H. Kernel software for efficiently building, reconfiguring, and distributing an open CNC controller. Int J Adv Manuf Technol 2006;27(7–8):788–96.
[11] Jöhnson A, Wall J, Broman G. A virtual machine concept for real-time simulation of machine tool dynamics. Int J Mach Tools Manuf 2005; 45(7–8):795–801.
[12] Wang Y, Liu T, Fu H, Han Z. Open architecture CNC system HITCNC and key technology. Chin J Mech Eng (English edition) 2007;20(2):13–6.
[13] Wang L, Brennan RW, Balasubramanian S, Norrie DH. Realizing holonic control with function blocks. Integrated Comput Aided Eng 2001;8(1):81–93.
[14] Gordon S, Hillery MT. Development of a high-speed CNC cutting machine using linear motors. J Mater Process Technol 2005;166(3):321–9.
[15] Chang Y-F. DSP-based ignition delay monitor and control of an electro-discharge machining process. Intell Automat Soft Comput 2007;13(2):139–51.
[16] Erkorkmaz K, Wong W. Rapid identification technique for virtual CNC drives. Int J Mach Tools Manuf 2007;47(9):1381–92.
[17] Wang L, Jin W, Feng HY. Embedding machining features in function blocks for distributed process planning. Int J Comput Integrated Manuf 2006;19(5): 443–52.
[18] Wang L, Hao Q, Shen W. A novel function block based integration approach to process planning and scheduling with execution control. Int J Manuf Technol Manage 2007;11(2):228–50.
[19] FBDK—Function Block Development Kit. Online: 〈www.holobloc.com〉, access date: December 2007.
[20] Christensen J. Design patterns for systems engineering with IEC 61499, Verteilte Automatisierung-Modelle und Methoden für Entwurf, Verifikation, Engineering und Instrumentierung, Ch. Döschner, ed. Magdeburg. Germany: Otto-von-Guericke-Universität; 2000.
[21] Vyatkin V. IEC 61499 function blocks for embedded control systems design: Instrumentation Society of America, 2007.
[22] RXTX Java Library. Online: 〈http://users.frii.com/jarvi/rxtx/index.html〉, accessed January 2008.
[23] ISO 14649-11. Data model for computerized numerical controllers: part 11–process data for milling. International Standards Organization; 2003.