PICARD on RT-Linux: a component software architecture for the real-time control system

Osok Song, Yunho Jeon, Dong-Young Kim, and Chong-Ho Choi
School of Electrical Engineering, Seoul National University, Korea
{osok, yunho, young, chchoi}@csl.snu.ac.kr

Abstract

This paper presents an architecture and design methods for rapid development of real-time control systems such as CNC or robot controllers. PICARD (Port-Interface Component Architecture for Real-time system Design) is a software architecture and environment which is aimed to reduce development time and cost of real-time control systems. With PICARD, a control engineer can construct a control system software by assembling pre-built software components using interactive graphical development environment. PICARD has been used to implement a prototype PC-based CNC controller successfully. PICARD consists of PICARD Virtual Machine(PVM), a component library, and PICARD Configuration Editor(PICE). PVM is a real-time engine of the PICARD system which runs control tasks on a real-time operating system. The component library is composed of components which are called as taskblocks. PICE is a visual editor which can configure control tasks by creating data-flow diagrams of taskblocks or ladder diagrams for sequential logics. PICARD has been implemented on several operating systems including QNX, WindowsNT (non-real-time), Linux (non-real-time), INtime, and RT-Linux. Compared to other operating systems, there are some differences in RT-Linux implementation in which real-time tasks should be written as kernel modules. For the communication between a PVM on a target system and PICE on a host computer, a simple protocol was devised because RPC or CORBA cannot be used in RT-Linux. By using description of communication functions in the Python language, function stubs are automatically generated. Experiences on using the C++ language in kernel modules are also discussed.

1 Introduction

Thanks to the rapid improvement of the PC hardware and real-time operating systems on PCs, the adoption of PCs for real-time systems, which traditionally used dedicated hardware devices, is becoming more common. For example, the controllers of CNC (Computerized Numerical Control) machines which handle user interaction while generating trajectories and controlling multiple servo motors are being replaced with PCs with real-time operating system.

While using PC hardware for such tasks reduces hardware cost, the cost increase due to difficulties of building reliable and efficient software can offset this advantage. The use of a real-time operating system only provides a basic means to build real-time software. Although there have been extensive studies on real-time system design methodology or software engineering, building robust and complex real-time software is still difficult even for experienced software engineers. What makes the problems worse is that parts of control system software are often written and maintained by control or mechanical engineers, who usually have neither enough knowledge nor experience on real-time system issues such as priority inversion, race condition or deadlock.

In order to facilitate designing control systems and maintaining their related software, PICARD (Port-Interface Component Architecture for Real-time system Design) has been developed. It is a software architecture and environment for rapid development of control systems such as CNC or robot controllers, which are traditionally implemented with PLCs (Programmable Logic Controllers), DSPs (Digital Signal Processors) and proprietary embedded controllers. The aim of PICARD is to make it easy to create efficient and reliable hard real-time software, using pre-built software components.

Software components are units of software which can be reused without modifying or recompiling the source code of the software. While a few widely used software component standards exist, they are not suitable for hard real-time systems. Software com-

ponents for real-time systems should be efficient and deterministic. They should also provide mechanisms to deal with problems such as race conditions, priority inversions and to predict and limit the maximum blocking time of time-critical task. PICARD solves this problem by limiting what a component is allowed to do at run-time and by exposing the data flow information between components to the configuration system. The configuration system can apply various validation and optimization methods without intervention of a human by using the information on the data flow between components as well as the information of the components. An interactive, graphical editor has been also developed to assist rapid development of control systems.
PICARD has been implemented on several operating systems, including WindowsNT (non-real-time), Linux (non-real-time), QNX, INtime, and RT-Linux [1]. Among these, RT-Linux shows good hard real-timeness and provides enough features which are required for PICARD such as networking and dynamic object file loading.
The rest of this paper is structured as follows: Section 2 is about the reviews of previous related studies. Section 3 introduces the architecture and some unique features of PICARD system. Section 4 is about issues on implementing PICARD on RT-Linux. Section 5 presents example systems which is built by using PICARD. Finally, concluding remarks are made in section 6.

2 Related Works

2.1 Component software

The component software technology is closely related to the object-oriented programming and is often mentioned without clear distinction. While both technologies use modularity (or encapsulation) to increase re-usability, component software emphasizes the ability to be deployed independently using well-defined, easily accessible interface. Many of the widely used component software standards, such as COM from Microsoft and JavaBean from Sun Microsystems, offer mechanisms to retrieve information on a component's interface without access to its source code or additional documentation. Combining this property with implementation techniques that allow dynamic loading of pre-compiled software modules, it becomes possible for third-party vendors to supply independently developed components without worrying that their proprietary algorithm might be disclosed.
One drawback of using component software is that it introduces overhead both in size and speed. This becomes a more important issue for control applications where memory size and processor speed are limited. In order to use the component software for control applications, the components should be efficient and have deterministic behavior. Furthermore, there should be a mechanism to manage access to shared resources from multiple tasks with minimal and deterministic influence on real-time performance. Traditional methods such as the priority ceiling protocol (PCP) or immediate priority ceiling protocol (IPCP) [2, 3] cannot be used efficiently because the PCP requires information about the entire system which is not available when each component is developed and compiled. For these reasons, the standard component models are not adequate for real-time control applications.

2.2 Models of computation

An important issue in component software for control systems is the selection of the model of computation. Many different models of computation for a control system have been used and studied. One of the popular approaches is the data-flow model. Represented by objects and (endless) streams of data flowing between them, the data-flow model looks similar to a hardware system familiar to many control engineers. This model has been successfully used in many application domains such as simulation (e.g. Simulink from The Math-Works), signal processing (e.g. Ptolemy: [4, 5]), data acquisition and analysis (e.g. LabView from National Instruments) and scientific visualization (e.g. Visualization Data Explorer from IBM), to name a few. The data-flow model is usually used in a visual programming environment and has an advantage for control systems with tight real-time requirements because the model reveals sensor-to-actuator data paths explicitly. Hence, the data-flow model is used in PICARD to decompose a control system function into multiple components and construct a control system using them.

Another popular model used for control systems is Finite State Machines (FSMs) and its variations. FSM is well suited to describe and analyze complex sequential control logic, but compared to the data-flow model, it is more difficult to deal with hard real-time requirements of feedback controllers. Thus, it is used together with the data-flow model in some control systems (e.g. [6]).

In the following, software methodologies closely related to PICARD are briefly introduced.

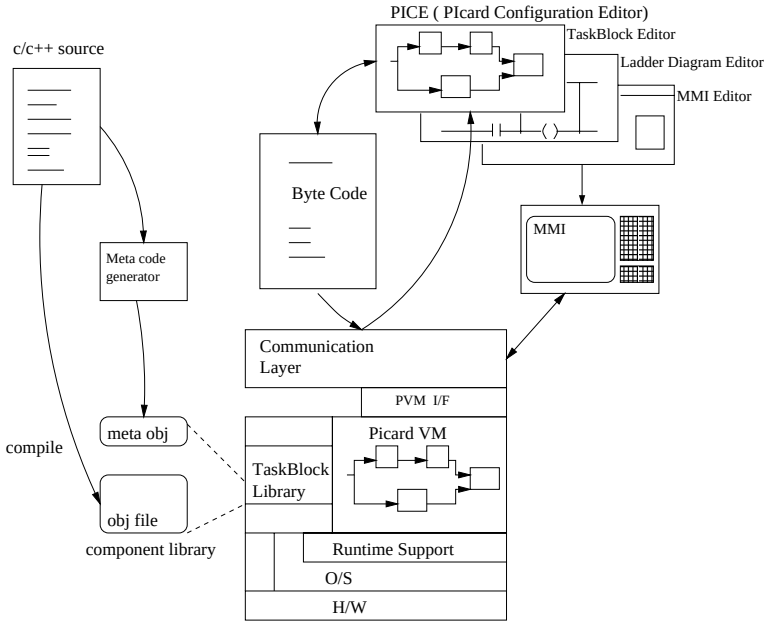


Figure 1: The overall architecture of PICARD

2.3 IEC 61131-3 standard

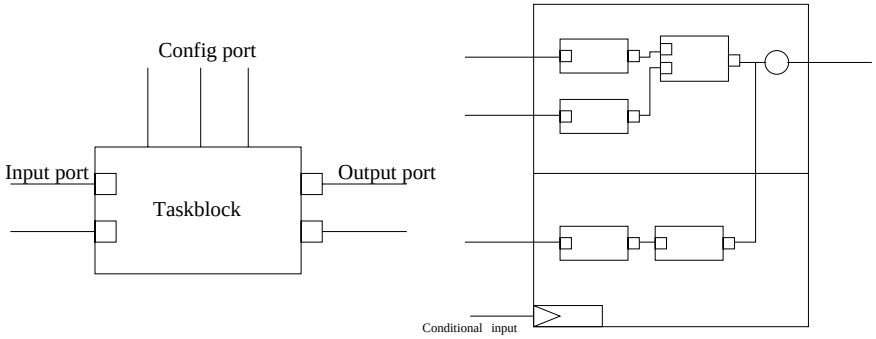
The IEC 61131-3 standard [7] defines common elements and programming languages for programmable logic controllers (PLCs) or similar systems in industrial automation.
The Function Block Diagram (FBD) is one of the programming languages defined in the standard. The FBD is a graphical programming language which looks like electronic circuit diagrams. Function blocks can be thought of as software components which can be used in FBD, though no standard distribution format is defined by the IEC 61131-3 standard. This representation is very natural when one thinks of function blocks as ICs (Integrated Circuits) or other electrical hardware components familiar to control engineers. Compared to the Ladder Diagram (LD), another graphical programming language primarily used to process binary values, the FBD deals with other data types in a more general way. Using the FBD, a control system can be constructed by connecting function blocks.

2.4 Chimera

The Chimera [8, 9, 10] is a real-time operating system developed to control reconfigurable robots. The Chimera Methodology is proposed as a solution to create dynamically reconfigurable real-time software using components which communicate using their ports. Similarly to function blocks of IEC 61131-3, port-based objects (PBO) of Chimera communicate with the outside only through their well-defined ports. PBOs are configured and connected to form a control system. Each PBO is an independent concurrent process that can execute at any frequency. One of the key features of Chimera is the way PBOs communicate with each other. In Chimera, each PBO keeps its local state variable table, which the PBO can access without synchronization. There is also a global state variable table, which is a union of all local tables. Consistency between the global and local tables is maintained by copying the contents in the critical section at the beginning and the end of each period. Because access to the global table is synchronized by a single global lock [8] or by disabling interrupts [9] there is no possibility of deadlock or priority inversion. It is assumed that the size of the data to copy in each period is not large in most control software so that copying them as a whole in a critical section does not affect real-time performance of the system.

3 PICARD Architecture

Three main components of PICARD are taskblocks, PICARD Virtual Machine(PVM)and PICARD Configuration Editor(PICE). PVM is a real-time engine of the PICARD system which executes control tasks on a real-time operating system. The component library is composed of components which are called taskblocks. PICE is a visual editor which a system designer can use to configure control tasks by creating data-flow diagrams of taskblocks or ladder diagrams. PICE includes an editor to compose Java-



(a) Taskblock (b) Conditional block

Figure 2: Graphical representation of taskblocks

based MMI program. Figure 1 shows the overall architecture of PICARD's implementation.

3.1 Taskblock

Taskblock is a basic software component in PICARD. Figure 2 (a) shows a graphical representation of a taskblock. It interacts with other taskblocks through well-defined interfaces called *port*s. Three port types are defined: input port, output port and configuration port. Input and output ports are used to transfer values between taskblocks. Configuration ports are used to set various operation parameters such as number of axes of machine tools and gains of feedback controllers. A taskblock class is implemented as a C++ class inherited from a common base class and all ports of taskblock can be used as normal variables in member functions.
The properties of a taskblock and its ports are described in a taskblock interface description file which is C++ header file with extended macros. It is parsed by the meta code generator to create meta code automatically. The meta code is compiled and linked together with the code that describes the behavior of the taskblock. The information of the taskblock such as names of ports, their data types and HTML documentations can be extracted and used by the run-time system when a system designer designs a control system. An example of taskblock description file is depicted below.

```
#include "basicTB.h"
class taskblock : public class BasicTB
{
  PICARD_TASK_BLOCK taskblock
         (doc="sample taskblock");
  PICARD_INPUT_PORT<double> input
         (doc="input port", default=0);
  PICARD_OUTPUT_PORT<double> output
         (doc="output port", default=0);
public:
  virtual int ready();
  virtual int fire();
}
PICARD_FILE "*.html", "*.gif"
```

At least two functions should be defined in taskblock.

- `ready()`:

  The actions which should be performed before a control system starts to run is defined in a `ready()` function such as variables and hardware initialization.

- `fire()`:

  When a taskblock is executed, this function of the taskblock is called in every period. A behavior which can cause blocking is not allowed in the `fire()` function. For example, if a taskblock is required to perform a file or network I/O, it should use asynchronous API to open and read the device and poll it in each period.

One or more taskblock binary files are linked to build a *taskblock library*. According to the operating system on which PVM is running, the taskblock libraries are statically linked to PVM or dynamically loaded.
There are other kinds of taskblocks which are implemented with PICARD bytecode*, instead of using taskblock classes and building binary files. They appear as taskblocks in PICE and are used in the same way as ordinary taskblocks when a system is designed. They are:

*PICARD bytecode will be explained in section 3.2.4

- *Composite taskblock*: A composite taskblock is an encapsulation of one or more taskblocks and connections between their ports. It is used as a taskblock and avoids repetition of the same configuration.

- *Conditional taskblock*: A conditional taskblock in PICARD corresponds to 'if' or 'switch' statements of textual programming languages. Using a conditional taskblock, groups of taskblocks can be activated or deactivated according to the value of a conditional port. Chimera achieves similar effect by allowing real-time switching between multiple configurations, and IEC 61131-3 provides a state programming language for this purpose. In PICARD, the change of configuration caused by a conditional taskblock is more localized to a small part of the control system. Figure 2 (b) shows an example of a conditional taskblock.

- *Delay taskblock*: A delay taskblock is a special taskblock which is assumed to have no temporal dependency between its input and output ports when the execution order is determined. At run time, a value presented at its input port is transfered to its output port in the next period. A delay taskblock is used for resolving loops in a data-flow of a taskblock diagram.

- *Inline taskblock*: Taskblocks which have simple run-time behavior can be constructed using PICARD bytecode. For example, taskblocks for adding two integers and for converting a real number to an integer are directly converted to bytecodes, not using taskblock objects. They allow to make taskblocks with simple algorithm without compiling C++ source code.

- *Queue block*: While most tasks in a control system are periodic in nature or can be processed periodically using polling, the periodic transfer of a fixed amount of data between tasks of different periods is often inadequate. In a CNC controller for machine tools, for example, a G-code interpreter and an interpolator typically run in a task with a relatively long period, while a position control taskblock runs in a task with a short period. If only a fixed amount of data can be transferred between the interpolator task and the control task, then the interpolator has to run at the same frequency as the control task, or the control task should have a buffering mechanism. Because situations like this are quite common in control applications, a *queue* is provided as a general solution.

  The internal code of the taskblock can examine the amount of space left in the queue and push or pop data as much as is required in a single period. A queue can be connected to multiple output ports of taskblocks running with different periods, but only one input port of a taskblock can be connected to it. A queue in PICARD is similar to multiple-input, single-output queues which are provided by many operating systems. A major difference is that a queue in PICARD does not cause the calling task (or thread) to be blocked to limit the execution time of the taskblock which attempts to put data to the queue.

3.2 PVM

PVM is a base platform for real-time task execution. It is responsible for loading taskblock libraries, communicating with PICE and MMI, running real-time tasks, and synchronizing data between real-time tasks. PVM is designed as simple as possible with minimal functionality for run-time execution so that it can fit into a small embedded system. All complex algorithms such as checking consistency of a configuration of a system and optimization methods are in PICE.

3.2.1 Execution of a task

A task is a set of one or more taskblocks and internal connections between their ports. A task is mapped to a process or a thread of the underlying operating system. Tasks are executed periodically and have fixed priorities which are determined by rate monotonic scheduling policy (RMS) [11, 12].
Taskblocks contained in a task are executed sequentially in each period, and the order of execution is determined by their topological order, as in IEC 61131-3, to minimize the latency from sensors to actuators. The following is a brief introduction of some unique features of PVM.

3.2.2 LVT/GVT Coherence Protocol

Values of all ports in a task are stored in the task's local variable table (LVT). To make an output port accessible to other tasks, the port should be *export*ed. When a port is exported, it occupies an entry in the global variable table (GVT) and can be accessed from other tasks by *import*ing it. Only one task can export (write) a port, but more than one task can import (read) it. The contents of LVT and the GVT should be kept coherent so that a change of value in an LVT entry can be propagated to corresponding entries of other LVTs. When a task begins execution,

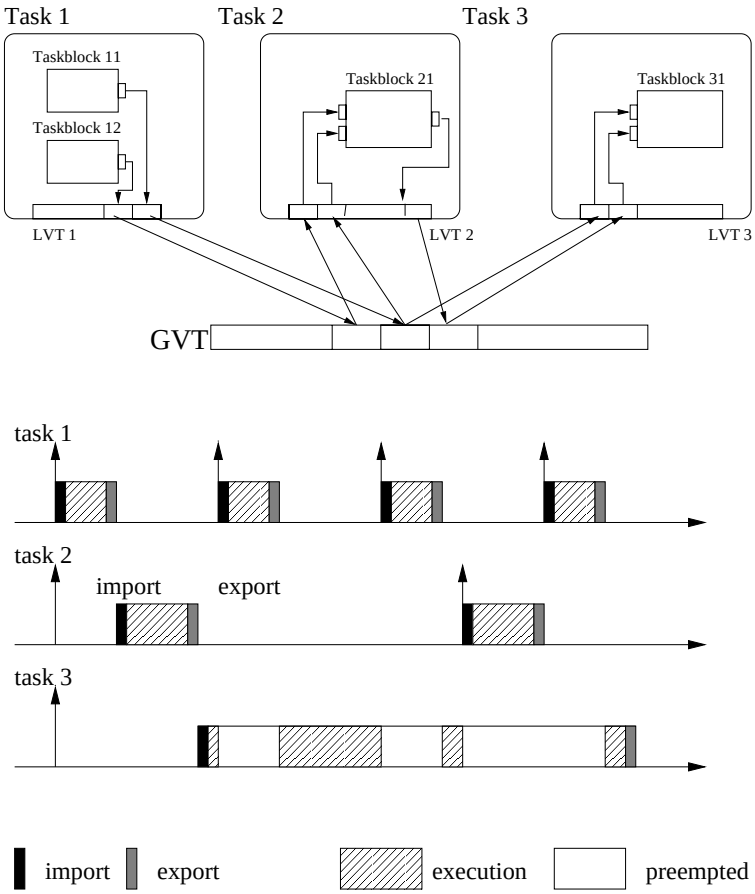


Figure 3: LVT/GVT coherence protocol

the values in the GVT corresponding to imported ports of the task are copied to the related LVT. At the end of the execution, the values of the LVT corresponding to exported ports are copied back to the GVT. Figure 3 shows example of a simple system which is consisted of three tasks. The up arrows of the lower part indicate release of each tasks.

The copying operation is executed in a critical section (by disabling interrupts), which are marked black in the figure, so that the variable tables always maintain consistency by preventing the copying operation from being preempted by higher-priority tasks. This protocol is similar to the communication method of Chimera. One difference is that in PICARD, taskblocks running at the same frequency are grouped as a task and the contents of all ports which are exported by a task are copied all at once in a critical section to maintain consistency.

One drawback of this protocol is that the blocking time due to the critical section increases as the number of import/export ports of a task increases. Methods to reduce this time is discussed in following section.

3.2.3 Minimizing jitters

As a control system gets more complicated and the number of variables shared between tasks increases, it takes more time to synchronize LVT/GVT. The performance of a time-critical task in a control system such as a high-speed servo motor control task can be affected by scheduling jitter due to the blocking from the synchronization of LVT/GVT.
Based on the idea of priority ceiling protocol, the following methods can reduce the maximum blocking time of a system using the data flow information between components [13]. Simplified description of the algorithm for the export ports is as follows.

At configuration time in PICE:

1. Determine priority ceiling values of export ports.
2. Sort and group the export ports of the task by their priority ceiling values in ascending order when the location of export ports are allocated in the shared memory.

At run time in PVM:

1. At the end of each period, copy values of export ports from LVT to GVT , of which priority ceiling value is less or equal to the priority of the task, P_i.
2. For copying the next group of export ports of which priority ceiling value is higher than P_i, raise P_i to priority ceiling value of the group of export ports and copy the data of that group from LVT to GVT.
3. Reset P_i to the original value and reschedule tasks.

The same procedure is also used for import ports. With this algorithm, the blocking time of high priority task can be significantly reduced. This method can be easily applied in the PICARD system in which the usage of shared memory of tasks is known to the configuration system.

3.2.4 PICARD bytecode

The run-time behavior of a task is defined by the PICARD bytecode. The bytecode is generated by the configuration system in PICE and transfered into PVM. The set of PICARD bytecode includes *RUNTB* for calling taskblock's **fire()** function, *COND_JUMP* for implementation of conditional block, and memory management commands such as *READ_N* and *WRITE_N*.
Bytecodes for changing priority of a current task and for calling the OS scheduler to reschedule the task set. These bytecodes are used for implementing jitter minimization algorithm.
In the bytecodes, the addresses are stored as handles. Before executing the bytecodes for the first time, handles are converted to real addresses by preprocessing process.

3.3 PICE

PICE is a graphical editor on the non real-time host computer. It is programmed with Java language. It consists of three editors with different functionalities.

3.3.1 Taskblock diagram editor

The taskblock diagram editor is used to compose a periodic task by connecting pre-built taskblocks as shown in Figure 4. When PICE starts to run, it is connected to a PVM via network (TCP/IP, serial bus, etc.), and get the the meta information of taskblocks from PVM. A control designer can examine the properties and documentations of taskblocks with property window.
A task is represented as a white workspace. By drag-and-drop of taskblocks and connecting ports of taskblocks, run-time behavior of the task is easily configured.

3.3.2 Ladder diagram editor

PICE provides another way to design and describe a task as shown in Figure 5. The ladder diagram editor used a LD language defined in IEC61131-3 standard. Besides the taskblocks for functions and function blocks defined in IEC61131-3 standard are provided, normal Taskblocks can be also integrated into the ladder diagram seamlessly. Variables in a ladder diagram can be exported and imported to and from GVT to communicate with other tasks in the system. A LD program is interpreted to PICARD bytecode and executed in the same way as tasks described with taskblock diagram. This editor can be used for constructing PLC task which can replace PLC based on proprietary hardware.

3.3.3 MMI editor

Figure 6 shows the MMI editor for developing Java MMI program of control system. The MMI editor provides JavaBeans components for monitoring and controlling system as the form of JavaBeans. These components include input components reading data from variables in PVM such as switches and output components writing data into variable in PVM such as gauges and graph.
Constructed MMI program can be executed as a Java applet or a Java application. If a target system on which PVM is running has a HTTP daemon program, the MMI program of the form of Java applet can be stored in a target system and a web browser can be used for executing MMI.

4 Implementing on RT-Linux

First version of PICARD was developed on Linux and QNX. For its good real-time performance and functionalities, RT-Linux was chosen as the main development platform of PICARD. There are some differences on RT-Linux implementation from other implementations because real-time tasks should be written as kernel modules.

4.1 Communication between PVM and PICE

A simple protocol was devised for the communication between PVM and PICE. There are two ways for communication: synchronous and asynchronous. Synchronous communication is a blocking function

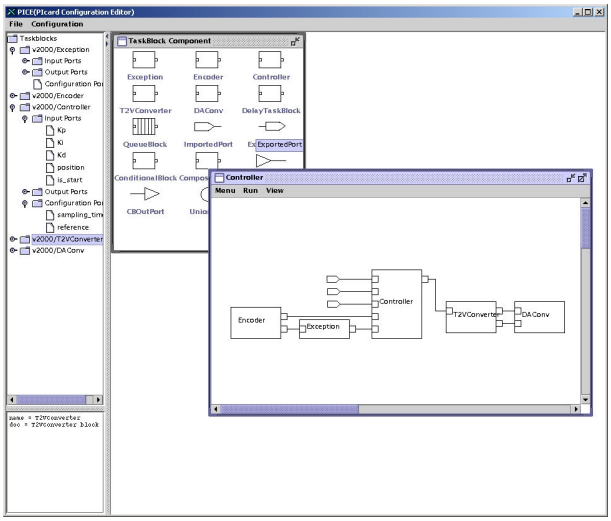


Figure 4: The Taskblock diagram editor of PICE

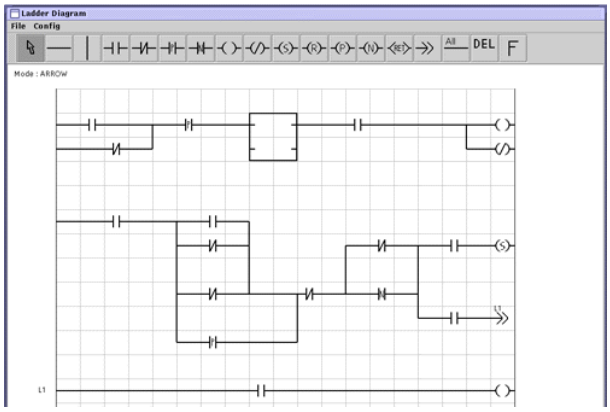


Figure 5: The ladder diagram editor of PICE

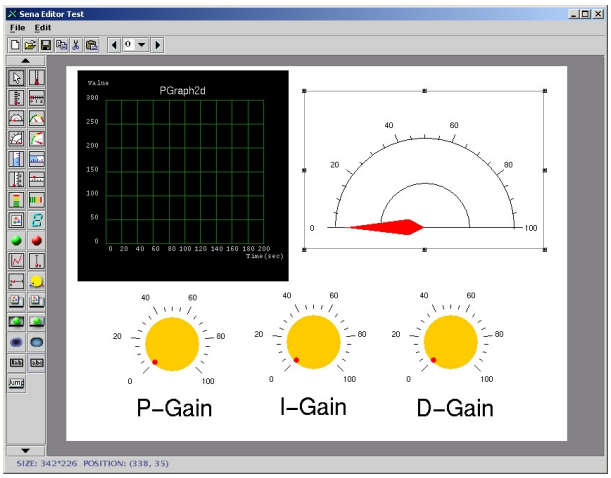


Figure 6: The MMI editor of PICE

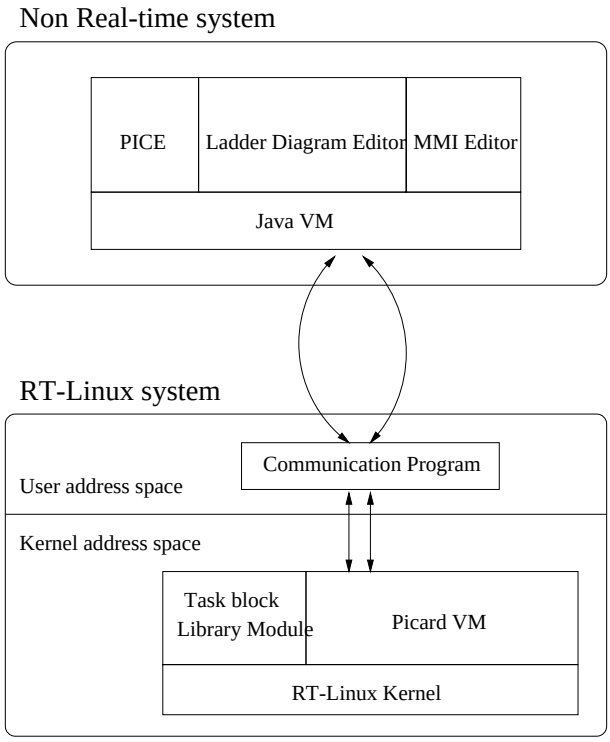


Figure 7: PICARD on RT-Linux

call to PVM from PICE or MMI program. The communication functions are described in Python language as follows.

```
API("readMem",          # name
    [Handle("hMemory"), Int32("offset"),
    Int32("size"), Int32("num")],
                        # arguments
    StringZ("data"))
                        # return value
```

The API generation tool parses this description and creates function stub files both for PICE in the Java language and for PVM in the C++ language.

Asynchronous communication is used for tasks in PVM to send data to PICE or MMI program at runtime. This method can be used for real-time data logging and error reporting.

In RT-Linux, PVM is divided into two programs as shown in Figure 7. Communication program in user address system is responsible for receiving and sending data from and to programs in a host computer. Synchronous communication between user space program and real-time part of PVM is done via device file while asynchronous one uses real-time FIFO which is provided by RT-Linux.

4.2 Implementation of jitter minimization method

To apply jitter minimization method in section 3.2.3 effectively, the time for changing task's priority should be very short. For operating systems such as QNX, for example, changing priority of a task involves context switching and takes tens of microseconds, which makes this method inefficient.
In RT-Linux, the current task's priority can be increased simply by writing a new priority value into the task's context data structure directly. This method is very efficient for RT-Linux or similar operating systems in which an application program can change a task's priority quickly.

4.3 The problem related to using C++

In RT-Linux, a real-time program should be written as a kernel module. Normally, C language is used to write a kernel module, but C++ language was used to implement PVM and taskblock libraries of PICARD.
To use C++ for creating kernel modules, several problems should be solved.

- **Global object construction and destruction**

When the linker and the startup code does not support C++, constructors of global variables are not called. There are several solutions for this problem, such as dynamic allocation of all objects and explicit call of constructor by using a 'placement new' operator[14]. The simplest one is linking C++ startup and exit code to the kernel module, then calling functions for global object construction and destruction, in `init_module()` and `cleanup_module()` respectively. which are `__do_global_ctors_aux()` and `__do_global_dtors_aux()` in case of `g++`.

- **Exporting functions of taskblock**

  In RT-Linux, a taskblock library is loaded to the RT-Linux kernel as a kernel module after PVM is loaded. Each taskblock in a library has a set of interface functions, and these functions are called by PVM to get the meta information of the taskblock, initialize and execute taskblock. The interface functions of taskblocks should be registered in PVM when they are loaded to the kernel, which is the member functions of C++ classes.

  Because the kernel module cannot export the member functions of C++ class directly, wrapper functions in standard C are exported and registered at a function table in PVM.

The operator `new` and `delete` should be defined explicitly as well to use kernel memory management functions.

5 Example : CNC controller

By using PICARD, PC-based CNC controller which is named SNU-NC has been built. To develop SNU-NC, the functionalities of CNC controller was decomposed to several tasks: main task for interpreting G-code and tool-path generation, control task and PLC task. Taskblocks for each task are also designed and developed. These taskblocks have well-defined interface so they can be reused for developing different controllers for other systems. SNU-NC successfully replaced the existing proprietary CNC controller and proved the effectiveness of PICARD. The taskblock diagram of main task and MMI are shown in Figure 8 and Figure 9 respectively.

6 Conclusion

In this paper, PICARD, a component software architecture for control systems is introduced. By using software components which provide well-defined interface and run-time accessible documentation, the effort for the development of real-time control system and maintenance is reduced.
PICARD is implemented on several operating systems including RT-Linux. For implementing PICARD on RT-Linux, several modifications were required. Most of them were related to converting user address space programs to kernel modules.
RT-Linux was chosen as the main development platform of PICARD. It is because RT-Linux shows good performance and provides enough features to implement PICARD.

References

[1] M. Barabanov, "A linux-based real-time operating system," M.S. thesis, New Mexico Institute of Mining and Technology, 1997.

[2] R. Rajkumar, L. Sha, and J.P Lehoczky, "An experimental investigation of synchronisation protocols," *Proceedings of 6th IEEE Workshop on Real-Time Operating Systems and Software*, 1989, pp. 11–17.

[3] L. Sha, R. Rajkumar, and J. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization," *IEEE Transactions on Software Engineering*, pp. 1175–1185, 1990.

[4] J. T. Buck, S. Ha, Lee E. A., and D. G. Messerschmitt, "Ptolemy: A framework for simulating and prototyping heterogeneous systems," *Int. Journal of Computer Simulation*, pp. 155–182, 1994.

[5] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, "Synthesis of embedded software from synchronous dataflow specifications," *Journal of VLSI Signal Processing Systems*, 1999.

[6] S. A. Schneider, V. W. Chen, G. Pardo-Castellote, and H. H. Wang, "Controlshell: A software architecture for complex electromechanical systems," *The International Journal of Robotics Research*, pp. 360–380, 1998.

[7] *International Standard IEC 1131-3, Part 3. Programming languages*, IEC, 1993.

[8] D. B. Stewart, R. Volpe, and P. Khosla, "Design of dynamically reconfigurable real-time software using port-based objects," *IEEE Trans. on Software Engineering*, pp. 759–776, 1997.

[9] M. Hassani and D. B. Stewart, "A mechanism for communicating in dynamically reconfigurable embedded systems," *Proc. High Assurance Software Engineering Workshop*, 1997, pp. 215–220.

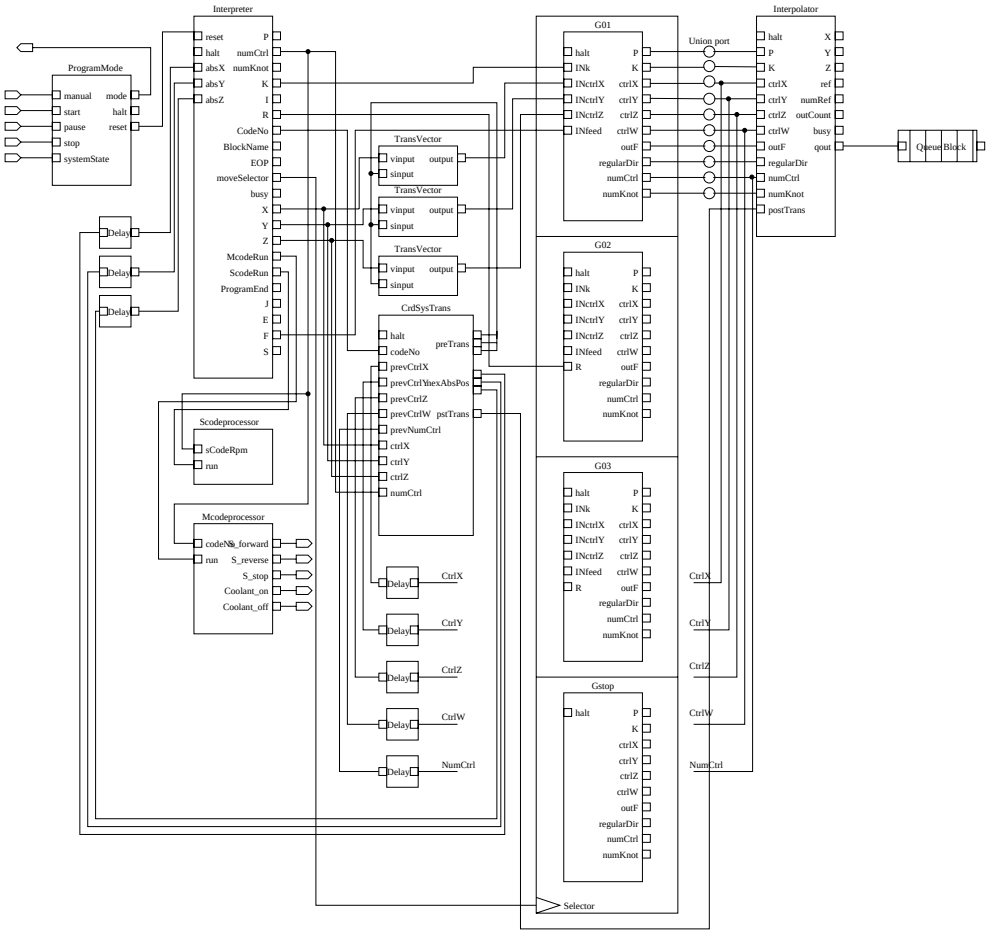


Figure 8: The Taskblock diagram of main task in SNU-NC

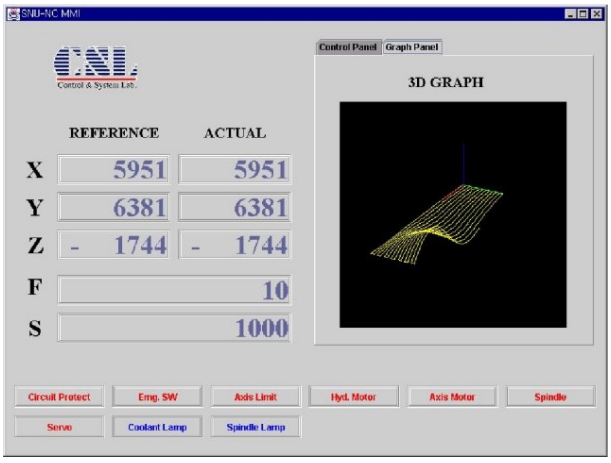


Figure 9: The MMI of SNU-NC

[10] D. B. Stewart, "Designing software components for real-time applications," *Embedded Systems Conference*, 1999.

[11] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the Association for Computing Machinery*, pp. 46–61, 1973.

[12] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: Exact characterization and average case behavior," *Proc. IEEE Real-Time Sys. Symp.*, 1989, pp. 166–171.

[13] Yunho Jeon and Chong-Ho Choi, "Minimization of blocking time in componenent-based software architecture for control systems," *Control Engineering Practice, submitted for publication.*

[14] B. Stroustruup, *The C++ programming language, third edition*, Addison-Wesley, 1997.