The Formal Design Model of a Real-Time Operating System (RTOS+): Conceptual and Architectural Frameworks

Yingxu Wang, University of Calgary, Canada
Cyprian F. Ngolah, Sentinel Trending & Diagnostics Ltd., Canada
Guangping Zeng, University of Science and Technology Beijing, China and University of California, Berkeley, USA
Philip C.-Y. Sheu, Wuhan University, China and University of California, Irvine, USA
C. Philip Choy, University of Calgary, Canada, Yousheng Tian, University of Calgary, Canada

ABSTRACT

A real-time operating system (RTOS) provides a platform for the design and implementation of a wide range of applications in real-time systems, embedded systems, and mission-critical systems. This paper presents a formal design model for a general RTOS known as RTOS+ that enables a specific target RTOS to be rigorously and efficiently derived in real-world applications. The methodology of a denotational mathematics, Real-Time Process Algebra (RTPA), is described for formally modeling and refining architectures, static behaviors, and dynamic behaviors of RTOS+. The conceptual model of the RTOS+ system is introduced as the initial requirements for the system. The architectural model of RTOS+ is created using RTPA architectural modeling methodologies and refined by a set of Unified Data Models (UDMs). The static behaviors of RTOS+ are specified and refined by a set of Unified Process Models (UPMs). The dynamic behaviors of the RTOS+ system are specified and refined by the real-time process scheduler and system dispatcher. This work is presented in two papers; the conceptual and architectural models of RTOS+ is described in this paper, while the static and dynamic behavioral models of RTOS+ will be elaborated in a forthcoming paper.


INTRODUCTION

An operating system is a set of integrated system software that organizes, manages, and controls the resources and computing power of a computer or a computer network. It also provides users a logical interface for accessing the physical machine in order to run applications. A general-purpose operating system may...
be perceived as an agent between the hardware and resources of a computer and the applications and users. An operating system may be divided into three subsystems known as those of the kernel or system management, the resource management, and the process management (Dijkstra, 1968; Brinch-Hansen, 1971; Liu & Layland, 1973; Peterson & Silberschatz, 1985; Anderson et al., 1989; McDermid, 1991; Deitel & Kogan, 1992; Tanenbaum, 1994; Viscarola & Mason, 2001; Silberschatz et al., 2003; Wang, 2004, 2007). The kernel of an operating system is a set of central components for computing, including CPU scheduling and process management. The resource management subsystem is a set of supporting functions for various system resources and user interfaces. The process management subsystem is a set of transition manipulation mechanisms for processes and threads interacting with the system kernel and resources.

A real-time operating system (RTOS) is an operating system that supports and guarantees timely responses to external and internal events of real-time systems (Laplante, 1977; Sha et al., 1990; ISO/IEC, 1996; Ford, 1997; Bollella et al., 2002; Kreuzinger et al., 2002; Ngolah, Wang, & Tan, 2004). An RTOS monitors, responds to, and controls an external environment, which is connected to the computer system through sensors, actuators, or other input-output (I/O) devices. In a real-time system in general and an RTOS in particular, the correctness of system behaviors depends not only on the logical results of computation but also on the time point at which the results are obtained. Real-time systems can be divided into hard and soft real-time systems. In the former, a failure to meet timing constraints will be of serious consequences, while in the latter, a timing failure may not significantly affect the functioning of the system.

A great variety of RTOS’s have been developed in the last decades (Lewis & Berg, 1998; Labrosse, 1999; Yodaiken, 1999; Rivas & Harbour, 2001; Lamie, 2008; ETTX, 2009). The existing RTOS’s are characterized as target-machine-specific, implementation-dependent, and not formally modeled. Therefore, they are usually not portable as a generic real-time operating system to be seamlessly incorporated into real-time or embedded system implementations. Problems often faced by RTOS’s are CPU and tasks scheduling, time/event management, and resource management. Efficient and precise methodologies and notations for describing solutions to these problems are critical in RTOS design and implementation. Generally, RTOS’s require multitasking, process threads, and a sufficient number of interrupt levels to deal with the random interrupt mechanisms in real-time systems. In modern RTOS’s, multitasking is a technique used for enabling multiple tasks to share a single processor. A thread is individual execution of a process in order to handle concurrent tasks. In addition, an interrupt is a request of an external device or internal process that causes the operating system to suspend the execution of a current low priority task, serve the interrupt, and hand control back to the interrupted process.

This paper develops a comprehensive design paradigm of the formal real-time operating system (RTOS+) in a top-down approach on the basis of the RTPA methodology. The conceptual model, architectural model, and the static/dynamic behavioral models of RTOS+ are systematically presented. In the remainder of this paper, the conceptual model of RTOS+ is described as the initial requirements for the system. The architectural model of RTOS+ is created based on the conceptual model using the RTPA architectural modeling methodologies and refined by a set of unified data models (UDMs). Then, the static behaviors of RTOS+ are specified and refined by a set of unified process models (UPMs). The dynamic behaviors of RTOS+ are specified and refined by process priority allocation, process scheduling, and system dispatching models. Due to its excessive length and complexity, this paper presents the first part of RTOS+ on its conceptual and architectural models, followed in serial by another paper that presents the second part of
this work on the static and dynamic behavioral models of RTOS+ (Wang et al., 2010c).

THE CONCEPTUAL MODEL OF RTOS+

The multi-task and multithread real-time operating system (RTOS+) is a portable and rigorous RTOS with formal architectural models and behavioral processes. As a preparation, this section presents the conceptual models of RTOS+ from the facets of its architectures, resources, and processes (tasks), as well as their interactions. The top-level framework and the process scheduling scheme of RTOS+ are informally introduced.

The Architecture of RTOS+

RTOS+ is designed for handling event-, time-, and interrupt-driven processes as a portable real-time system platform. The conceptual architecture of RTOS+ is described as shown in Figure 1, where interactions between system resources, components, and internal control models are illustrated. The conceptual architecture of RTOS+ can be divided into three subsystems:  

a) the processor and the CPU/process/system schedulers, b) the resource/event controllers, and c) the process control structures.

The schedulers for CPU, processes, and the entire system are the innermost operating system kernel that directly controls the CPU as well as system tasks, processes, events, and resources. The process scheduling mechanism of RTOS+ is priority-based. A flexible priority scheduling technique is adopted in RTOS+, where the priority of a given process for a task is assigned based on its type and importance when it is created. Processes are categorized into five priority levels known as the high and low periodic interrupts, device I/O interrupt, user application interrupts, and the base levels. A process, when it is created, will be put into a proper queue corresponding to its predefined priority level.

The resources of RTOS+ are modeled by the entities of memory, time, devices (ports), interrupt facilities, files, and internal control structures such as dispatching queues, system/device/event/memory, and process control blocks. The resources in RTOS+ can be classified into three categories known as the system resources (memory, device, and files), event

Figure 1. The architecture of RTOS+
resources (system timing events, interrupts, and I/O events), and resource control structures (running processes and tasks in other states).

**Process Scheduling in RTOS+**

The process scheduling of RTOS+ is priority-based and event-driven as shown in Figure 2. The process scheduler is responsible for determining which process should be executed on the processor at a given time interval of CPU time. The dynamic performance of tasks in RTOS+ is embodied by a series of coherent processes and threads. A task is created by the system (State 1) as a process in the waiting queue with an assigned priority based on its importance and timing requirement. When the required resource is available, the process pending in the waiting queue will be scheduled into a ready queue with the proper priority. The process scheduler dispatches tasks in the high ready queues before scheduling those in the low ready queue in State 2. It continuously checks the ready queues to see if there is any process ready to be run in State 3. If there are ready processes in the queue, it compares the priorities of the currently executing process and that of the first process in the queue. If the first process in the queue has a higher priority, it pre-empts the CPU from the currently running process, assigns the CPU to the higher priority process from the queue, and sends the lower priority process to the delayed queue in State 6. The delayed process is re-dispatched to the appropriate ready queue to be executed when the CPU becomes available. If there are two processes of the same priority, the process scheduler treats them on a first-come-first-serve basis.

A scheduled process executes on the CPU until it is either completed (State 4) or suspended due to lack of a resource or pending for an event (State 7). As illustrated in Figure 2, there are three conditions that may cause a running process to be re-scheduled out of running: a) Interrupted by a process or event with higher priority; b) Timed-out for a pre-scheduled time-interval of CPU; and c) Waiting for a specific device or event. An interrupted running

*Figure 2. State transition diagram of the process scheduler in RTOS+*
process is sent to the interrupted queue (State 5) and later returned to the appropriate ready queue when the interrupt service (State 9) is over. A process that has exhausted its assigned time interval must go to the delayed queue (State 6), while it can be re-scheduled back to one of the ready queues immediately for waiting for the next available CPU executing interval. However, a process that can no longer be executed due to lack of memory or resource has to be sent to the suspended queue (State 7), which may return to the appropriate ready queue when the pending resource has become available. A process interrupted, delayed, or suspended at States 5, 6, or 7, respectively, may be killed (State 8) by the process scheduler in case there is a persistent unavailability on demanded resources or CPU time.

More rigorous description of the system dispatching and process scheduling strategies and algorithms will be elaborated by the entire dynamic behaviors of RTOS+, particularly the CPUSchedulerST (Figure 18), ProcessSchedulerST (Figure 31), and SystemDispatcherST (Figure 30) from the bottom up. The following sections will extend and refine the top level framework of the RTOS+§ into a set of detailed architectural models (UDMs) and behavioral process models (UPMs).

**THE ARCHITECTURAL MODEL OF THE RTOS+ SYSTEM**

The architecture of a real-time system is a framework that represents the overall structure, components, processes, as well as their interrelationships and interactions. This section formally specifies the architecture of RTOS+, RTOS+§. ArchitectureST, based on its conceptual models as provided in the preceding section (Figures 1-2). Each of the architectural components will be modeled and refined as a UDM (also known as component logical model (CLM)) (Wang, 2002, 2008a).

**The Architectural Framework of RTOS+**

System architectures, at the top level, specify a list of identifiers of UDMs and their relations. A UDM may be regarded as a predefined class of system hardware or internal control models, which can be inherited or implemented by corresponding UDM objects as specific instances in the succeeding architectural refinement for the system.

Corresponding to the conceptual model of RTOS+ as shown in Figures 1 and 2, the high-level specification of the architecture of RTOS+, RTOS+§.ArchitectureST, is given in Fig. 3. RTOS+§.ArchitectureST encompasses parallel structures of three architectural subsystems known as ProcessorST, ResourcesST, and ProcessesST, as well as a set of events @EventsS and a set of statuses @StatusesBL. Each subsystem may be further refined by detailed UDMs as shown in Figure 3, where the numbers in the angle brackets indicate the configuration of a specific UDM in the system. Detailed UDMs for all structural components in the three subsystems will be specified and refined in the following subsections.

\[
\text{§(RTOS+)} \triangleq \text{RTOS+§.ArchitectureST} \\
\quad \| \text{RTOS+§.StaticBehaviorsPC} \\
\quad \| \text{RTOS+§.DynamicBehaviorsPC}
\]

(1)

where || indicates that these three subsystems are related in parallel, and §, ST, and PC are type suffixes of system, system structure, and process, respectively.
The Architecture of the Processor Subsystem

According to the high level architectural model of RTOS+ as given in Fig. 3, the processor subsystem of RTOS+, ProcessorST, represents an abstract model of the system kernel such as the CPUST, SystemControlBlockST, EventControlBlockST, and SystemClockST. The following subsections formally describe the architectures of the kernel components by a set of UDMs in RTPA.

The CPU

The CPU of RTOS+ is formally modeled as a set of parallel computing entities with their associated resources in a broad view. The CPU's computing capabilities are logically abstracted as a set of processes and underlying resources such as CPU time, processes, and the event-/time-/interrupt-driven dispatch mechanisms; while the computing resources are modeled by the system clock, memory, devices (ports), system variables, and system status.

An abstract model of the target CPU of RTOS+ is described as shown in Figure 4, where the CPUST is modeled as a generic computing system (Wang, 2007), §, which is a virtual machine of the executing platform of the processor denoted by a set of parallel computing resources and processes. The computing system § controls all computing resources of the abstract target machine and the behavioral processes embodied on them.

The System Control Block

The system control block (SCB) is the central and top-level control structure of the entire RTOS+. The UDM of SCB, SystemControlBlockST = SCBST, is modeled as shown in Figure 5. SCBST specifies a set of system constants such as the maximum capacities of processes (threads), memory, devices (ports), events, interrupts, and timers. Then, the current numbers of dynamic...
process and resource usages of system are modeled within the constraints of system capacity constants. The numbers of the current running process (PN), event (EN), and device (DN) are specially modeled for system control.

**The Event Control Blocks**

An event control block (ECB) is a global control structure for system event management. The UDM of ECBs, EventControlBlockST = \( \text{ECB}(\text{EN}) \), is modeled as shown in Figure 6, which is a dynamic structure with varying number of ECBs, controlled by SCBST #EventsN, as a series of indexed records. An ECBST registers the key informant of an event in RTOS+ such as its status, type, time detected, device required, and source/target process (if any).

**The System Clock**

A system clock is a typical real-time device for event timing, process duration manipulation, and system synchronization. The UDM model of the system clock of RTOS, SysClockST, is designed as given in Figure 7. SysClockST provides an absolute (calendar) clock \( \text{§t}_{hh:mm:ss:ms} \) as the logical time reference for the entire system and a relative clock \( \text{§t}_N \) as a generic counter. The real-time system clock will be updated at run-time by the process SysClockPC as shown in Figure 15.

**The Architecture of the Resources Subsystem**

According to the high level architectural model of RTOS+ as given in Figure 3, the resource subsystem of RTOS+, ResourcesST, manages...
system internal and external resources such as memory, devices (ports), and files. The following subsections formally describe the architectures of the resource components by a set of UDMs in RTPA.

**The Memory Control Blocks**

Memory management is one of the key functions of operating systems because memory is both the working space and storage of data or files.
Common memory management technologies of operating systems are contiguous allocation, paging, segmentation, and combinations of these methods (Silberschatz et al., 2003).

The abstract model of the memory system, MEM\textsubscript{ST}, can be described as follows:

$$\text{MEM}_{\text{ST}} \triangleq [\text{addr}_1] \ldots [\text{addr}_n] \mathbb{B} \ (2)$$

Typical memory manipulations are addressing, memory allocation, memory release, read, and write, as modeled in the RTPA meta-processes.

The addressing manipulation, ⇒, is a fundamental operation of RTPA that maps a given logical \(id_S\) into a block of the physical memory denoted by \(addr_P\) accommodating \(n\) bytes of memory, i.e.:

$$id_S \Rightarrow \text{MEM}[addr_P] \mathbb{B}$$
$$\Leftrightarrow (\pi: \text{id}_S \rightarrow \text{addr}_P)$$
$$\quad \Rightarrow \text{MEM}[addr_P, addr_P+nH-1] \mathbb{B} \) (3)

where \(\pi: \text{id}_S \rightarrow \text{addr}_P\) is a function that maps a given logical \(id_S\) into the physical memory block identified by \(addr_P\) in \(\text{MEM}[addr_P, addr_P+nH-1] \mathbb{B}\), in which \(nH\) is the size of the memory block and \(\mathbb{P}\) is a power set of \(P\).

Memory allocation, \(\Leftarrow\), is a fundamental computing operation that collects a unique memory block logically named \(id_S\) and physically located by \(addr_P\) accommodating \(n\) bytes of memory, i.e.:

$$id_S \Leftarrow \text{MEM}[addr_P] \mathbb{B}$$
$$\Leftrightarrow (\pi^{-1}: \text{addr}_P \rightarrow \text{id}_S)$$
$$\quad \Rightarrow \text{id}_S = \text{MEM}[addr_P, addr_P+nH-1] \mathbb{B} \) (4)

Memory allocation is a key meta-process for dynamic memory manipulation in RTOS+. The memory allocation process, \(id_S \Leftarrow \text{MEM}[addr_P, addr_P+nH-1] \mathbb{B}\), will be implemented in Figure 19.

Memory release, \(\Leftarrow\), is an inverse operation that dissociates and frees a unique block of \(n\) continuous physical memory elements denoted by \(addr_P\) from its logical identifier \(id_S\), i.e.:

$$id_S \Leftarrow \text{MEM}[\perp] \mathbb{B}$$
$$\Leftrightarrow (\pi: \text{id}_S \rightarrow \text{addr}_P$$
$$\quad \Rightarrow \text{MEM}[addr_P, addr_P+nH-1] \mathbb{B} := \perp)$$
$$\quad \Rightarrow addr_P := \perp$$
$$\quad \Rightarrow id_S := \perp \) (5)

\[\text{Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.}\]
Memory release is a key meta-process that dissociates a given memory block from a logical identifier id_S, and returns the memory block to the system. The memory release process, \( id_S \# MEM[\bot]B \), will be implemented in Figure 19.

Memory read, \( \triangleright \), is a meta-process of RTPA that gets data \( xB \) from a given memory location \( MEM[addr_P] \), where \( addr_P \) is a pointer that identifies the physical memory address, i.e.:

\[
MEM[addr_P]B \triangleright xB \tag{6}
\]

Memory write, \( \triangleleft \), is a meta-process of RTPA that puts data \( xB \) to a given memory location \( MEM[addr_P] \), where \( addr_P \) is a pointer that identifies the physical memory address, i.e.:

\[
xB \triangleleft MEM[addr_P]B \tag{7}
\]

The memory control block (MCB) is a global control structure for system memory management. The UDM of MCBs of RTOS+, \( MemoryControlBlockST = \sum^{\#MemBlocksUsed}_{MBN=1} R \), MCB(MBN\_ST), is modeled as shown in Figure 8, which is a dynamic structure with varying number of MCBs controlled by \( SCBST.\#MemBlocksUsedN \) as a series of indexed records. An MCB(MBN\_ST) registers the key information, such as its status, size in terms of a number of 10kB, starting address, time allocated, and required process (if any), into the allocated memory block identified by the memory block number (MBN\_N) in RTOS+.

The Device Control Block and Ports

Devices in an operating system encompass a variety of generic and special-purpose hardware and interfaces. Typical I/O devices that an operating system deals with are those of system, storage, human interface, communication, and special devices. I/O devices are connected to the computer through buses with specific ports or I/O addresses. Usually, between an I/O device and the bus, there is a device controller and an associated device driver. The I/O management system of an operating system is designed to enable users to use and access system I/O devices seamlessly, harmoniously, and efficiently. I/O management techniques of operating systems can be described as polling, interrupt, DMA, and socket.

A general abstraction of device interfaces can be modeled by a special system architectural type known as the port. The generic system I/O port model, PORT\_ST, can be described as a finite linear space, i.e.:

\[
PORT\_ST \triangleq [addr_1H \ldots addr_2H]B \tag{8}
\]

where \( addr_1H \) and \( addr_2H \) are the start and end addresses of the port space, and \( B \) is the byte type of each of the port I/O interfaces.

Typical port manipulations are port input and output as modeled in the RTPA meta-processes. An input, denoted by \( \triangleright \), is a metaprocess that receives data \( xB \) from a given system I/O port \( PORT[addr_P]B \), where \( addr_P \) is a pointer that identifies the physical address of the port interface, i.e.:

\[
xB \triangleright PORT[addr_P]B \tag{9}
\]

An output, denoted by \( \triangleleft \), is a meta-process that sends data \( xB \) to a given system I/O port \( PORT[addr_P]B \), where \( addr_P \) is a pointer that identifies the physical address of the port interface, i.e.:

\[
xB \triangleleft PORT[addr_P]B \tag{10}
\]

The device control block (DCB) is a global control structure for system device management. The UDM of DCBs of RTOS+, \( DeviceControlBlockST = \sum^{\#Devices}_{DN=1} R \), DCB(DN\_ST), is formally modeled in Figure 9, which is a dynamic structure with varying number of DCBs controlled by \( SCBST.\#DevicesN \) as a series of indexed records. A DCB(DN\_ST) registers the key informant, such as the device ID, type, status, addresses of input/
output, interrupt ports, time allocated, and required process (if any), into a corresponding DCB identified by the device number (DN) in RTOS+.

**Files**

The file system is a type of abstract resources of operating systems. A file is a logical storage unit of data or code separated from its physical implementation and location. Types of files can be text, source code, executable code, object code, word processor formatted, or system library code. The attributes of files can be identified by name, type, location (path of directory), size, date/time, user ID, and access control information. Logical file structures can be classified as sequential and random files. The former are files that organize information as a list of ordered records; while the latter are files with fixed-length logical records accessible by its block number.

The file system of an operating system consists of a set of files and a directory structure that organizes all files and provides detailed information about them. The major function of a file management system is to map logical files onto physical storage devices such as disks or tapes. Most file systems organize files by a tree-structured directory. A file in the file system can be identified by its name and detailed attributes provided by the file directory. The most frequently used method for directory management is a hash table. A physical file system can be implemented by contiguous, linked, and indexed allocation. Contiguous allocation can suffer from external fragmentation. Direct-access
is inefficient with linked allocation. Indexed allocation may require substantial overheads for its index block. Typical file operations are reading, writing, and appending. Common file management operations are creating, deleting, opening, closing, copying, and renaming.

The file system of RTOS+, FilesST, is modeled using an index table and a set of records as shown in Figure 10. In the UDM of FilesST, each record has two components known as the IndexST and RecordST. The former uses an index iN in the index table to find the key, KeyN, that points to a record in RecordST. The latter is a set of records that holds a specific data record to KeyN. The formal model of FilesST supports both sequential and random access to the file system of RTOS+. Using the primary key of a record and its address, a record can be randomly accessed; while for sequential access, the entries in the index table can be sequentially accessed to locate a particular record through its address.

The Architecture of the Processes Subsystem

According to the high level architectural model of RTOS+ as given in Figure 3, the process subsystem of RTOS+, ProcessesST, models a set of internal control UDMs and priority control queues such as the ProcessControlBlockST, RequestQST, Ready-QHST/ReadyQLST for high/low priority processes, CompletedQST, InterruptedQST, DelayedQST, SuspendedQST, which support system state transitions in task scheduling as shown in Figures 1 and 2. The following subsections formally describe the architectures of the process control components by a set of UDMs in RTPA.

The Process Control Blocks

The UDM of the process control block in RTOS+, ProcessControlBlocksST = \( R_{\text{SCBST} \# \text{Proc}} \), is specified in Figure 11. PCB(NN)ST, models multiple tasks in RTOS+ indexed by the process number PN. Each task in the system is created as a process with a process number, a task priority class, status, time created, and resources required for such as memory, device, and event.

The Process Dispatching Queues

As specified in SCBST as shown in Fig. 5, there are a set of nine dispatching queues adopted in RTOS+ such as RequestQST, WaitingQST, ReadyQHST, ReadyQLST, CompletedQST, InterruptedQST, DelayedQST, SuspendedQST, and KilledQST. A generic queue structure is
modeled as shown in Fig. 12 that is shared by all system queues where the instances of the DispatchingQueues\textsubscript{ST} are also refined with their initial values.

The behaviors of queues can be generically modeled as enqueue, serve, clear, empty test and full test in addition to create and release of queues (Wang, 2007), i.e.:

\[
\text{Queue}^{\text{ST}, \text{StaticBehaviors}_{\text{PC}}} \triangleq \begin{align*}
\text{Create}_{\text{PC}} \cup \\
\text{Enqueue}_{\text{PC}} \cup \\
\text{Serve}_{\text{PC}} \cup \\
\text{Clear}_{\text{PC}} \cup \\
\text{EmptyTest}_{\text{PC}} \cup \\
\text{FullTest}_{\text{PC}} \\
\text{Release}_{\text{PC}}
\end{align*}
\]

where enqueue is a process that appends a process at the end of the queue; and serve is a process that fetches the front process of the queue and shifts the remainder elements toward the front of the queue by one element.

The system architectural models specified in this section provide a set of abstract object models and rigorous interfaces between system hardware and software. By reaching this point, the co-design of a real-time system can be separately carried out by separated hardware and software teams. It is recognized that system architecture specification by the means of UDMs is a fundamental and the most difficult part in software system modeling, while con-
Conventional technologies and languages hardly provide any support for this purpose. On the basis of the system architecture specification and with the work products of system architectural components or UDMs, the specification of the operational components of the RTOS+ system as behavioral processes can be carried out directly as elaborated in the second part of this work on the static and dynamic behavioral models of RTOS+ as presented in (Wang et al., 2010c).

CONCLUSION

The design of real-time operating systems has been recognized as a comprehensive and complex system design paradigm in computing, software engineering, and information system design. This paper has demonstrated that the RTOS+ system, including its architecture, static behaviors, and dynamic behaviors, can be essentially and sufficiently described by RTPA. On the basis of the formal specifications of RTOS+ by the coherent set of UDMs and UPMs in RTPA, the architectural and behavioral consistency and run-time integrity of an RTOS have been significantly enhanced. It has been identified that RTOS+ can be applied not only as a formal design paradigm of RTOS’s, but also a support framework for a wide range of applications in design and implementation of real-time and embedded systems. The RTOS+ system model may have also provided a test bench for the expressive power and modeling capability of existing formal methods in software engineering.

With a stepwise specification and refinement methodology for describing both system architectural and operational components, the formal model of the RTOS+ system has provided a foundation for implementation of a derived real-time operating system in multiple programming languages and on different operating platforms. It has also improved the controllability, reliability, maintainability, and quality of the design and implementation in real-time software engineering. The formal models of RTOS+ have been adopted as part of the supporting environment for the implementation of the RTPA-based software code generator (RTPA-CG) (Wang et al., 2010b; Ngolah & Wang, 2009). On the basis of the formal and rigorous models of the RTOS+ system, code can be automatically generated by RTPA-CG or be manually transferred from the formal models.

A series of formal design models of real-world and real-time applications in RTPA have been developed using RTPA notations and methodologies (Wang, 2002, 2007, 2008a, 2008b, 2008c, 2009a; Wang & Huang, 2008) in the formal design engineering approach, such as the telephone switching system (Wang, 2009b), the lift dispatching system (Wang et al., 2009), the automated teller machine (ATM) (Wang et al., 2010a), the real-time operating system (RTOS+), the air traffic control system (to be reported), the railway dispatching system (to be reported), and the intelligent traffic lights control system (to be reported). Further studies have demonstrated that RTPA is not only useful as a generic notation and methodology for software engineering, but also good at modeling human cognitive processes in cognitive informatics and computational intelligence as reported in (Wang, 2008d, 2009a; Wang & Ruhe, 2007; Wang & Chiew, 2010).

ACKNOWLEDGMENT

The authors would like to acknowledge the Natural Science and Engineering Council of Canada (NSERC) for its partial support to this work. We would like to thank the reviewers for their valuable comments and suggestions.

REFERENCES


Yingxu Wang is professor of cognitive informatics and software engineering, Director of International Center for Cognitive Informatics (ICfCI), and Director of Theoretical and Empirical Software Engineering Research Center (TESERC) at the University of Calgary. He is a Fellow of WIF, a P.Eng of Canada, a Senior Member of IEEE and ACM, and a member of ISO/IEC JTC1 and the Canadian Advisory Committee (CAC) for ISO. He received a PhD in Software Engineering from The Nottingham Trent University, UK, in 1997, and a BSc in Electrical Engineering from Shanghai Tiedao University in 1983. He has industrial experience since 1972 and has been a full professor since 1994. He was a visiting professor in the Computing Laboratory at Oxford University in 1995, Dept. of Computer Science at Stanford University in 2008, and the Berkeley Initiative in Soft Computing (BISC) Lab at University of California, Berkeley in 2008, respectively. He is the founder and steering committee chair of the annual IEEE International Conference on Cognitive Informatics (ICCI). He is founding Editor-in-Chief of International Journal of Cognitive Informatics and Natural Intelligence (IJCINI), founding Editor-in-Chief of International Journal of Software Science and Computational Intelligence (IJSSCI), Associate Editor of IEEE Trans on System, Man, and Cybernetics (A), and Editor-in-Chief of CRC Book Series in Software Engineering. He is the initiator of a number of cutting-edge research fields and/or subject areas such as cognitive informatics, cognitive computing, abstract intelligence, denotational mathematics, theoretical software engineering, coordinative work organization theory, cognitive complexity of software, and built-in tests. He has published over 100 peer reviewed journal papers, 200+ peer reviewed conference papers, and 12 books in cognitive informatics, software engineering, and computational intelligence. He is the recipient of dozens international awards on academic leadership, outstanding contribution, research achievement, best paper, and teaching in the last 35 years.
Cyprian F. Ngolah received a PhD in Software Engineering from the University of Calgary, Canada in 2006, an MSc in Control Engineering from the University of Bradford, England in 1989, and a BSc in Mathematics and Computer Science from the University of Essex, England in 1988. He taught several computer science and software engineering courses at both the graduate and undergraduate levels for thirteen years at University of Buea, Cameroon. He is currently a senior software engineer in the Research and Development Department of Sentinel Trending & Diagnostics Ltd, Calgary, carrying out research on the development of a neural network for machine condition monitoring and predictive maintenance using vibration analysis. His main research interests are in real-time process algebra and its applications, tool support for formal specification languages, real-time operating systems, formal methods in software engineering and real-time software systems, and artificial neural networks.

Guangping Zeng is a professor of computer science and Director of School of Information Engineering at the University of Science and Technology, Beijing (USTB), China. He received a PhD and MSc in computer science from USTB in 2005 and 1995, respectively, and a BSc in Petroleum Engineering from the Chinese Petrol University in 1984. He was a postdoctoral fellow in EECS at University of California, Berkeley during 2008 to 2009. He was the former head of Department of Computer Science and Technology at USTB in 2003-2005. He is a senior member and a steering committee member of the Chinese Association of AI. He has published over 70 peer-reviewed papers and two books. His research interests are in artificial intelligence, intelligent software, soft-man, virtual robots, embedded networks, and Internet-based computing.

Phillip C-Y. Sheu is currently a professor of Computer Engineering, Information and Computer Science, and Biomedical Engineering at the University of California, Irvine. He received his PhD and MSc degrees from the University of California at Berkeley in Electrical Engineering and Computer Science in 1986 and 1982, respectively, and his BSc degree from National Taiwan University in Electrical Engineering in 1978. Between 1982 and 1986, he also worked as a computer scientist at Systems Control Technology, Inc., Palo Alto, CA., where he designed and implemented aircraft expert control systems, and he worked as a product planning engineer at Advanced Micro Devices Inc., Sunnyvale, CA, where he designed and integrated CAD systems. From 1986 to 1988, he was an assistant professor at School of Electrical Engineering, Purdue University. From 1989 to 1993, he was an associate professor of Electrical and Computer Engineering at Rutgers University. He has published two books: (1) Intelligent Robotic Planning Systems and (2) Software Engineering and Environment - An Object-Oriented Perspective, and more than 100 papers in object-relational data and knowledge engineering and their applications, and biomedical computations. He is currently active in research related to complex biological systems, knowledge-based medicine, semantic software engineering, proactive web technologies, and large real-time knowledge systems for defense and homeland security. His current research projects are sponsored by the National Science Foundation, National Institute of Health, and Department of Defense. Dr. Sheu is a Fellow of IEEE.

Philip Choy is a PhD candidate in Software Engineering with the Theoretical and Empirical Software Engineering Research Center (TESERC) at the University of Calgary. He received his BSc (Hon.) in Computer and Information Science from Queen's University, Kingston., Canada in 1985. He is a member of the IEEE and has served on the Canadian Board of the IEEE for over a decade. He has also held a number of executive positions with the IEEE Southern Alberta Section (SAS) including the Chairmanship during 2007 and 2008. He has over two decades of
experience in the IT, software, and telecommunications industries, where he has held various executive positions. His research interests are in software engineering, cognitive informatics, and real-time systems as well as their industrial applications.

Yousheng Tian is a PhD candidate in cognitive computing and software engineering with the International Center for Cognitive Informatics and Cognitive Computing (ICCICC) as well as Theoretical and Empirical Software Engineering Center (TESERC) in Dept. of Electrical and Computer Engineering at the University of Calgary, Canada. He received a PhD from Xian Jiantong University, China, in Computer Science in 2002 and was a Post Doctoral Fellow at ICCICC during 2006 to 2007. His research interests are in cognitive informatics, cognitive computing, software engineering, machine learning, and denotational mathematics.