Formal Specification and Verification of PLC for Certification*

Jin-Hyun Kim
Dept. of Computer Science,
Korea University
1, 5Ga, Anam-Dong,
Sunbuk-Gu
Seoul, 136-701, Korea
jhkim@formal.korea.ac.kr

Na Young Lee
Dept. of Nuclear Engineering
Seoul National University
56-1, Shillim-Dong
Gwanak-Gu,
Seoul, 151-742, Korea
grasia2@snu.ac.kr

Jin-Young Choi
Dept. of Computer Science,
Korea University
1, 5Ga, Anam-Dong,
Sunbuk-Gu
Seoul, 136-701, Korea
choi@formal.korea.ac.kr

ABSTRACT

KNICS (Korea Nuclear Instrumentation and Control System) is a national promoted project to develop a safety-critical level embedded system for nuclear plant protection system. PLC(Programmable Logic Controller) is a typical embedded system to instrument and control plant system, and KNICS has been developing a PLC for controlling a reactor of nuclear power plant system. The PLC micro-kernel is a safety-critical software that should be certified by KINS (Korea Institute of Nuclear Safety), the certification organization in Korea. In this paper, we present our experience on developing micro-kernel in PLC based on formal specification and formal verification. Using formal methods, we gain correctness of the target software and when the project ends, we will apply a certification to KINS.

1. INTRODUCTION

KNICS is a nationally promoted project of which goal is to develop a digitalized instrumentation control system technology for nuclear power plant. PLC is a critical embedded system to be developed in KNICS, which instruments and controls a reactor of nuclear power plant. The PLC will be certificated to the safety-critical level software, thus, it has been developed by rigorous development methods and process. The PLC micro-kernel in KNICS is a system software that manages temporal and spatial resources of software and hardware in the system. To gain a certification of the PLC to KINS, the certification organization in KOREA, we apply formal methods to specification and verification of the micro-kernel software.

In this work, we apply the resource model-based methodology to this micro-kernel development. The resource model-based development is where function and resource is separate from one another, and the resource model is a specific system model that specifies resource behavior over time and provides requirements for each of the micro-kernel functions.

*This work is supported by KNICS(Korea Nuclear Instrumentation and Control System

In this paper, we present our development process and development models for the PLC micro-kernel. In particular, we apply the resource model-based development to development of it.

This paper is organized as following. First, we present related works and introduce the development process for the PLC micro-kernel and the resource model-based method. Next, we informally explain the PLC micro-kernel software and the relevant entities in the system. And we provide formal model of the software model to verify the model formally. Finally, we conclude the paper.

2. RELATED WORKS

To develop micro-kernels in safety-critical systems such as avionics, automobile and nuclear power plant, applying formal methods has been studied rigorously now a days. Honeywell DEOS[12] is a real-time operating system that is developed with using formal methods. The DEOS is described in Promela(the input language for Spin), and Spin model checker[8] is used to verify time partitioning problem in its scheduling algorithm. During verification of it, a few already known error was rediscovered in time partitioning implementation.

In nuclear power plant systems, it is common to apply COTS (Commercial Off-The-Shelf) to certify RTOS kernels because there are over 100 commercial RTOSs, and there is no standard for the quality and the reliability of them. However, COTS is not suited for a newly developed embedded system because of a short development lifecycle. Moreover, there are no the golden-models for developing of real-time operating systems. In recent years, resource model is considered as an suitable model for embedded systems and real-time operating systems because most of the embedded systems is limited in using of resources.

Oleg Sololosky[13] gives some insight into resource in developing embedded system where resource is limited not only on quantity but also on quality. His research defines resource as processing power, memory, network bandwidth, and power consumption and proposes a uniform framework for a formal treatment of the resource by using PACSR(Probabilistic Algebra of Communicating Shared Resource) based on process algebra. However, his resource model is described in terms of quantitative aspect, not in the behavior of resource. Lee et.al[11] provides a resource model that is used for verifying a system in terms of timing constraints. And Choi[5]

provides a scheduling algorithm reasoning method for schedulability analysis based on Lee's research[11]. However, it is hard to identify resource models and function models from the system model so that it is not easy to implement the system.

During developing of the PLC software, we figure out that the resource in embedded systems has a specific and dynamic behavior, and provides the requirements for function behavior in forms of the behavior of the resource. In particular, the resource describing such dynamic behavior also gives timing constraints that functions in embedded systems must satisfy during operating of the system.

In the next chapter, we introduce micro-kernel models for certifying it to the regulation and guide in nuclear power plant. Next, we explain the development process for the system software for the PLC. And then, we gives the micro-kernel models of the PLC based on the resource models.

3. THE DEVELOPMENT PROCESS

IEEE 1012[2] requires development processes methods and V&V methods to be all differentiated according to the integrity level because the criticality of each software depends on the system. The PLC micro-kernel is safety-critical level software because it is used to control the core reactor of nuclear power plants. According to IEEE 1012[2], 830[3] and 1016[4], the requirement and design for the safety-critical level software must be described in mathematical notations and must be proved by mathematical proof techniques. In addition, we apply the V development model with using formal methods in specifying and verifying of the kernel software models; statecharts and model checker in STATEMATE MAGNUM(I-Logix).

Figure 1 shows the development process for the micro-kernel software of the PLC. To develop the requirement specification for the micro-kernel, we refer to IEEE 830[3] where general requirement contents for safety-critical systems are illustrated. To develop design specification, we refer to IEEE 1016[4] and IEEE 1016.1[1] that provide contents and development procedure for the design specification. IEEE 1012[2] explains the verifying and validating of software according to the system criticality. The gray-colored box in Figure 1 is where we apply the resource model-based method to micro-kernel software models.

Most systems in nuclear power plants require the highest reliability and safety recommended by IAEA(International Atomic Energy Agency). Therefore, the nuclear society are so conservative in applying new development methods and processes that only well-established proven methods and processes can be used for developing nuclear embedded systems. In this work, we adopt V-process model and formal CASE-tools for the PLC micro-kernel software. However, we can develop the resource based model in constructing requirement and design models for functions of the kernel models. The resource-view development methodology is called resource model-based methodology.

3.1 Resource Oriented Model for Embedded Systems

The behavior of most functions in software in embedded systems depends on hardware resources since it can accomplish its purpose while changing hardware resource state. The behavior of resource representing hardware captures the requirement for the relevant software, in particular, timing constraints over the software.

The resource models are divided into two classes; Synchronous-type and asynchronous-typed resource. Synchronous-typed resource is what directly interacts with software function. The resource provides inputs and its states to the function, the it takes some actions according the input and states. That is, the function and the resource behavior depends on each other. In contrast, asynchronous-typed resource affects function behavior in one-way, not being affected by function behavior. Thus, the asynchronous-typed resource changes its behavior by itself.

In addition, other characteristic of the resource model is concerning time. Most of the resources related to software functions can have timing constraints and the timing constraints is related to the functions, that is, the software functions also have timing constraints. Therefore, in this paper, we identify resource type relevant to timing constraints that is related to the hardware resource.

Figure 2 shows an example of a synchronous-typed resource where a semaphore synchronizes an interrupt handler and two tasks. One of two tasks named TASK 2 has timing constraints given by the semaphore so that the semaphore should put the task to a ready-list for scheduling unless it synchronizes the task by a certain time. When the semaphore receives a synchronizing signal from the interrupt handler, it should synchronize a task named TASK 1.

Figure 3 depicts the resource model-based development for developing the PLC micro-kernel. The system requirements in the figure are the user's requirement where products to be developed are described in natural languages. Next, it is necessary to identify necessary resources for accomplish the requirement. In this stage, the resources can be already-exiting resources that are already known

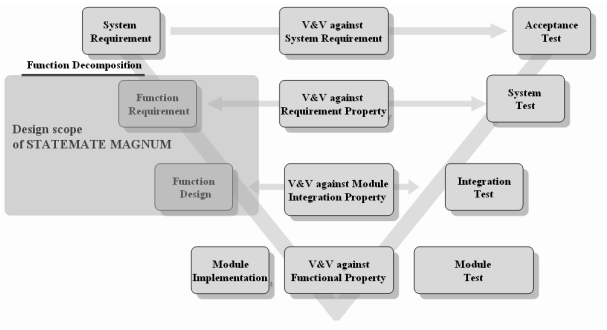


Figure 1: Development process for the micro-kernel of the PLC

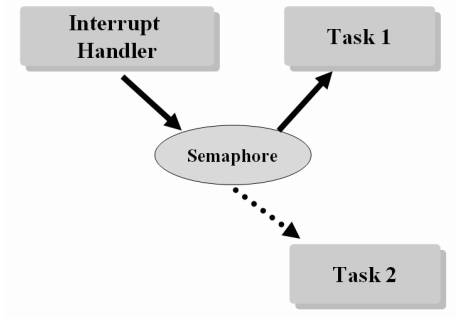


Figure 2: An example of the synchronous-typed resource

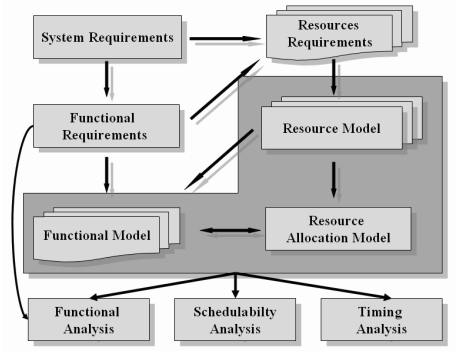


Figure 3: Resource model-based development

to developer. After that, newly developed functions would identified for user's requirement, then the resource identified at the previous stage are utilized to allow the functions to accomplish the requirement. At the same time, some resources are newly identified for the need of the functions. Then, resource models are constructed in forms of a behavioral description such automata, finite state-machine, state-diagram and so forth. After the modeling of resource behavior, each function model can be constructed with referring to behavior of the resource. The resource model provides useful information for the newly developed functions. That is, the resource model can provide information how the functions to be developed exploit the resource in the systems. For example, the resource model can capture timing requirements which functions to be developed shall satisfy in performing their programs. Thus, the function designer should construct function models with satisfying the timing requirements that the resource model provides. Next, it is needed to construct the resource assignment model for scheduling shared resources and verifying the system. The resource assignment model provides criteria to assign the resource to each function without failing to correctly execute the whole system and corrupting the resources.

In verification stages, a system is analyzed in terms of the functionality and timing. In addition, schedulability analysis is also included into the verification. In the functional analysis, it is checked whether it performs its function without fail even if each function is threaded with other functions. In the timing analysis, it is checked whether each of functions finish its performance by an appointed time, deadline. The schedulability analysis checked whether all of functions having timing constraints finish each of jobs by each deadline.

4. THE PLC MICRO-KERNEL

Micro-kernel of the PLC is what manages spatial and temporal resources by using protection mechanisms and scheduling policy in the system. To verify the micro-kernel system model, it is necessary to reason the behavior of the system whether the resources in the system are legitimately shared by software functions in terms of timing and behavior. A PLC is a special-purpose computer system that operates plant engineer's programs. The PLC provides five standard programming languages for a newly developed function; SFC, ST, IL, FBD and LD. And two kinds of heterogenous I/O devices can be installed into PLC in order to communicate with other systems.

Figure 4 describes an overview of software system in the PLC. After a program of a plant engineer is transformed into a machine code, it is transmitted into a free bank of memory. The function for the loading and the allocating is PES Agent. Display Agent and Diagnosis Agent are reporting the PLC status to users, and Sequence Solver is an user's special-purpose scheduler that instantly manipulates the order of tasks' running.

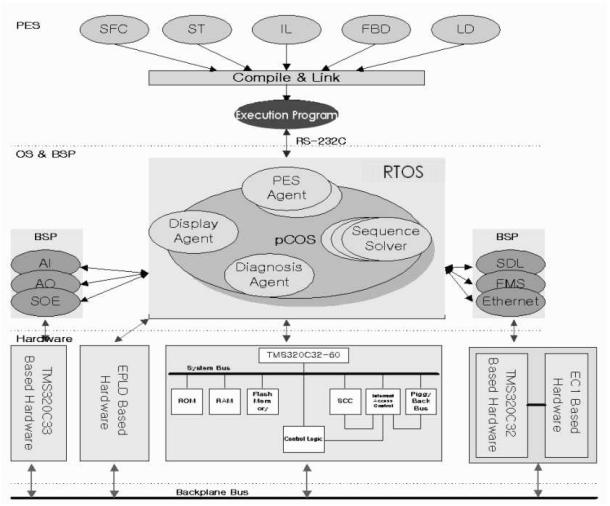


Figure 4: Micro-kernel software in the PLC

The micro-kernel for the PLC functions are more simple rather than ones that general-purpose real-time operating system provide. And the RTOS for safety-critical systems should be absolutely deterministic to guarantee the safety and reliability even if it is in any dangerous situations. Therefore, the functions are needed to be optimized for each use of the functions.

- Scheduling : The PLC micro-kernel gives a priority-based scheduling algorithm for real-time scheduling where any lower priority task cannot proceed its execution faster than the highest priority task. In the priority-based scheduling, any tasks must its execution by the next clock, that is, any task must not postpone performances until the next period.

- Inter-task communicating: The micro-kernel provides semaphore, message-queue, and message mailbox for inter-task communication. The semaphore is used to synchronize tasks. The message queue and mailbox are used to communicate data between tasks and synchronize tasks.

- Interrupt handling : Interrupt handling function is a special-typed task for handling an interrupt form environment. That is, the interrupt handler is almost same to general tasks except that it begins its execution by interrupt while the tasks begins by a scheduler.

- Task creating, deleting, idling and suspending : The micro-kernel provides functions for task creating, deleting, idling and suspending. When a task is created, the task acquires Task Control Block(TCB) that is used to control task's activation by a scheduler and is subscribed in ready-list for scheduling. When a task is deleted, the TCB relevant to the task is returned to the kernel system in order to have other new task acquire the TCB.

- Time managing : A task can request the kernel system to delay its execution for a certain amount of time. After the delay time, the kernel put the delayed task into ready-list for rescheduling.
- System idling and reporting : There are two system tasks in the kernel; idle and statistics task. The idle task is what takes over a CPU whenever any other tasks do not their execution any more. The statistics task diagnoses system status.

In addition, time-related functions exist to manage the system time, however, there are no memory managing system and no deadlock-related mechanisms that are essential to protect deadlock.

5. FORMAL SPECIFICATION

To apply the resource model-based method, firstly, the necessary resource is identified resources from system requirement. The resources in the PLC micro-kernel are as follows.

- Task Control Block(TCB) : TCB is a synchronous-typed resource for performance a application program in the PLC. It is controlled to preempt CPU by the micro-kernel scheduler, and the resource manager also used it for resource scheduling. The behavior of it is deterministic as having the following state; Dormant, ready, running, waiting and interrupt service routine.
- Inter-Task Communications (semaphore, message queue and message mailbox) : Most ITC elements are all synchronous-typed resources that are shared by tasks or interrupt handlers. All of them has deterministic behavior as follows; Used or not-used, and full, not-full or empty, and request or not-request. Each of resources can have timing constraints.
- Memory : Memory is a synchronous-typed resource that is physically shared by processes and interrupt handlers, hence it should not be corrupted by any illegal accesses of processes and interrupts handlers.
- I/O port : I/O port is shared by the environment and internal functions in the system. And it may have timing constraints to synchronize the environment and software functions. I/O port can implemented in synchronous-typed or asynchronous typed resource according to I/O mechanisms.

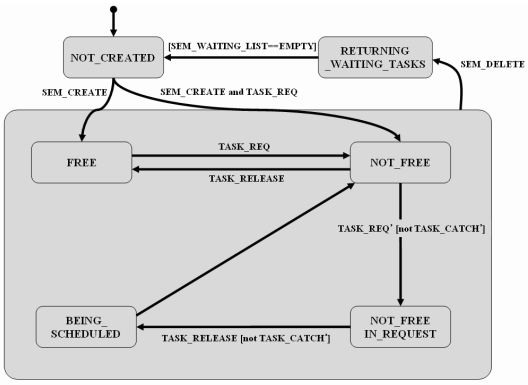


Figure 5: Semaphore behavior model

Figure 5 shows a simple semaphore behavior depicted in statecharts(The syntax and semantics of the statecharts in this paper can be referred to [6] and [7]). The states that the semaphore has in the system are as follows; NOT_CREATED, FREE, NOT_FREE, NOT_FREE, IN_REQUEST and BEING_SCHEDULED. The requirement for the semaphore can be explained as follows.

- A semaphore can be created by a task. Then, the state of the semaphore is determined by the creating task;
- A task can acquire a semaphore only when the semaphore is not acquired by any other tasks.
- A task can request a semaphore even if the semaphore is already acquired by other tasks, and the task can wait for the semaphore releasing. If the semaphore is released, it is scheduled according to task priority, and the highest priority task can acquire it.

The model of a semaphore in Figure 5 depicts only the resource behavior that will be used by any tasks. The following requirements are more specific description for the semaphore functions that is used by tasks. The following functions are described with focusing on the semaphore resource behavior.

- SEM_CREATE : SEM_CREATE creates a semaphore, and a task invoking the SEM_CREATE immediately can preempt it, or can just create it without preempting semaphore.
- SEM_PEND : A task invoking SEM_PEND can acquire a semaphore only when the semaphore is not acquired by any other tasks. Otherwise, the task should move from the ready-list to the semaphore-requesting list.
- SEM_POST : A task invoking SEM_POST releases a semaphore. If there exist other tasks waiting for the semaphore, they should be scheduled to acquire it according to task priority.
- SEM_DELETE : SEM_DELETE deletes a existing semaphore. When a semaphore is deleted, the relevant semaphore-requesting list should also deleted, and the tasks waiting the semaphore should be return to the ready-list.

Functional models for the micro-kernel are constructed with referring to resource models that is created previously. Figure 6 shows each behavior of semaphore-related functions in statecharts. All of functions related semaphores should consider all of the states of the semaphore because the resource type of the semaphore is synchronous-typed. For instance, the semaphore-creating function can create a semaphore only when the semaphore is already not created. In addition, The functions related to the semaphore resource should consider all of the states of the resource. In other words, all of the resource states must be satisfied by behavior of all semaphore-using functions.

The semaphore functions is implemented with based on the function models.(Figure 7).

Now, we explain the models for the PLC micro-kernel. Figure 8(a) shows a TCB resource that represents that task has DORMANT, WAIT, READY, SYSTEM_CALL, RUNNING and ISR state, and

task representing both TCB and user's program executes user's program when TCB has RUNNING state. All of the transition in the TCB model depends on scheduling and ITC functions. Message mailbox resource(Figure 8(b)) is more complicated than the TCB model. It shows all of resource behavior related to functions and depicts that when and how the functions utilize the resource in detail. Figure 9(a) shows a scheduling functions where the task are scheduled according to the fixed-priority scheduling policy. The mark ">" indicates that there exits some internal behavior in the state so a behavior of a function can task place in the state. Figure 9(b) shows the semaphore-creating function, which is more complicated rather than that in Figure 6(a). It disables the interrupt that comes from the environment in order to protect a shared resource(Event Control Block) to be corrupted.

6. FORMAL VERIFICATION

There are two kinds of formal verification methods in STATEMATE MAGNUM. First, ModelChecker[10] is a simple verifier to check whether the model is satisfied in terms of non-determinism of transitions, race-condition of simultaneous read/write, reachability to all or one state and so on. It gives the engineer a simple and easy way for checking the consistency, completeness and robustness of models. Another verifier is ModelCertifier[9], in which verification property can be expressed in forms of predefined temporal property patterns. The user knowing the semantics of these patterns, the user can define very complex properties easily by instantiating pattern parameters with STATEMATE MAGNUM expressions.

To apply model checking to the kernel models, firstly, we perform ModelChecker to check the syntactical correctness. Moreover, we perform reachability analysis to check completeness of the model of the system. As shown in the functional models(Figure 6), they have final state where indicates function's success. We can easily check the completeness by checking whether the final state in the function model can be reachable from the beginning state of the function. After that, we perform model checking to verify the user's requirement and property by using ModelCertifier.

The first verification property is mutual exclusion, and it is described as follows.

More than two tasks never shares a semaphore.

A task can invoke SEM_PEND function while other task invokes another SEM_PEND function simultaneously. Then, only one of SEM_PEND functions can enter into the SUCCESS_SEM_ACQUIRE state that indicates that a task acquires the semaphore resource. The property for the mutual exclusion in formal language is as follows. In this paper, we use the temporal logic of ModelCertifier in STATEMATE MAGNUM(I-Logix).

in(TASK_1:SEM_PEND:SUCCESS_SEM_ACQUIRE)
and
in(TASK_2:SEM_PEND:SUCCESS_SEM_ACQUIRE)

The verification for mutual exclusion results in the micro-kernel model satisfying the property.

Now, we are applying model checking for such properties as schedulability, priority inversion, and mutual exclusion.

7. CONCLUSION

In this paper, we introduced formal models of a micro-kernel for PLC to be developed in KNICS. The PLC is a safety-critical level embedded system that should be certified by the certification organization in Korea.

For specification and verification of the model of the PLC micro-kernel, we use statecharts and model checking methods in STATEMATME MAGNUM.

When constructing the models, we apply the resource model-based development methods to spiflication of the micro-kernel model because the resource in embedded systems plays a critical role in terms of shares and time. Hence, we depicts dynamic behavior of the resource for software functions, and verify the whole system based on the resource behavior. In result, the function model for each of the micro-kernel functions is specified with centering around the resource identified from the requirement, and the verification is easily performed by checking the resource states.

In future, we will verify more specific verification properties by using model checking and develop software requirement and design models relatively based on the resource models.

8. REFERENCES

[1] IEEE std. 1016.1, guide to software design description, 1993.

[2] IEEE std 1012, standard for software verification and validation, 1998.

[3] IEEE std. 803, recommended practice for software requirement specification, 1998.

[4] IEEE std.1016, recommended practice for software design specification, 1998.

[5] J.-Y. Choi, I. Lee, and H. liang Xie. The specification and schedulability analysis of real-time systems using ACSR. In *IEEE Real-Time Systems Symposium*, pages 266–275, 1995.

[6] D. Harel. Statecharts: A visual formalism for complex systems. *Sci. Comput. Program.*, 8(3):231–274, 1987.

[7] D. Harel and A. Naamad. The statemate semantics of statecharts. *ACM Trans. Softw. Eng. Methodol.*, 5(4):293–333, 1996.

[8] G. Holzmann. *The SPIN MODEL CHECKER*. Addison-Wesley, September 2003.

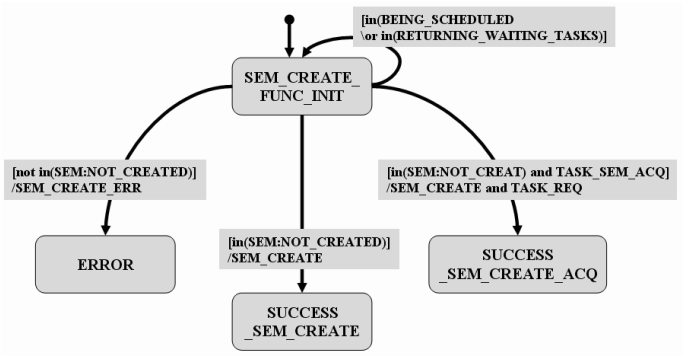
[9] http://www.osc es.de/products/en/modelcertifier.php.

[10] http://www.osc es.de/products/en/modelchecker.php.

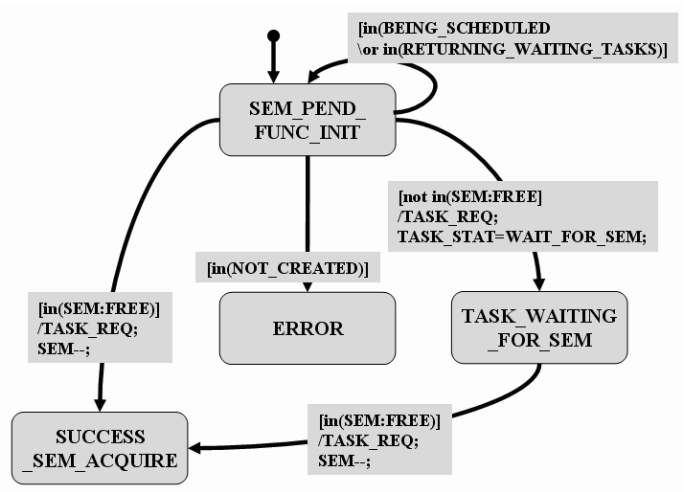
[11] I. Lee, P. Br'emond-Gr'egoire, and R. Gerber. A process algebraic approach to the specification and analysis of resource-bound real-time systems, 1994.

[12] J. Penix, W. Visser, E. Engstrom, A. Larson, and N. Weininger. Verification of time partitioning in the deos scheduler kernel. In *ICSE '00: Proceedings of the 22nd international conference on Software engineering*, pages 488–497, New York, NY, USA, 2000. ACM Press.

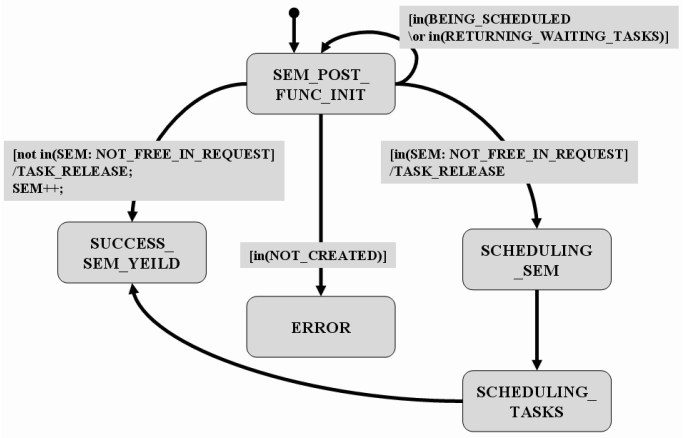
[13] O. Sokolsky. Resource modeling for embedded systems design. In *WSTFEUS*, pages 99–103, 2004.



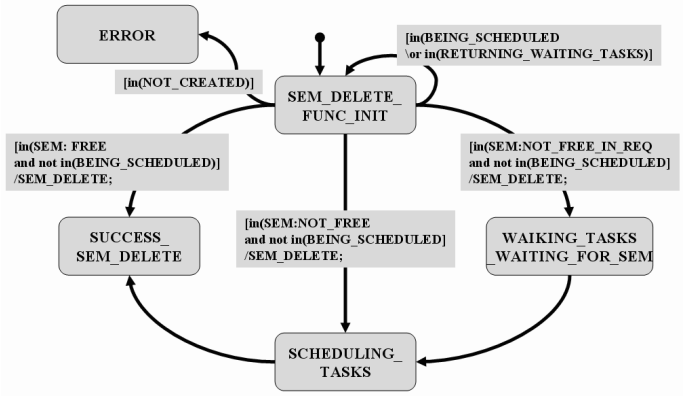
(a) Creating function



(b) Pending function



(c) Posting function



(d) Deleting function

Figure 6: Semaphore function models

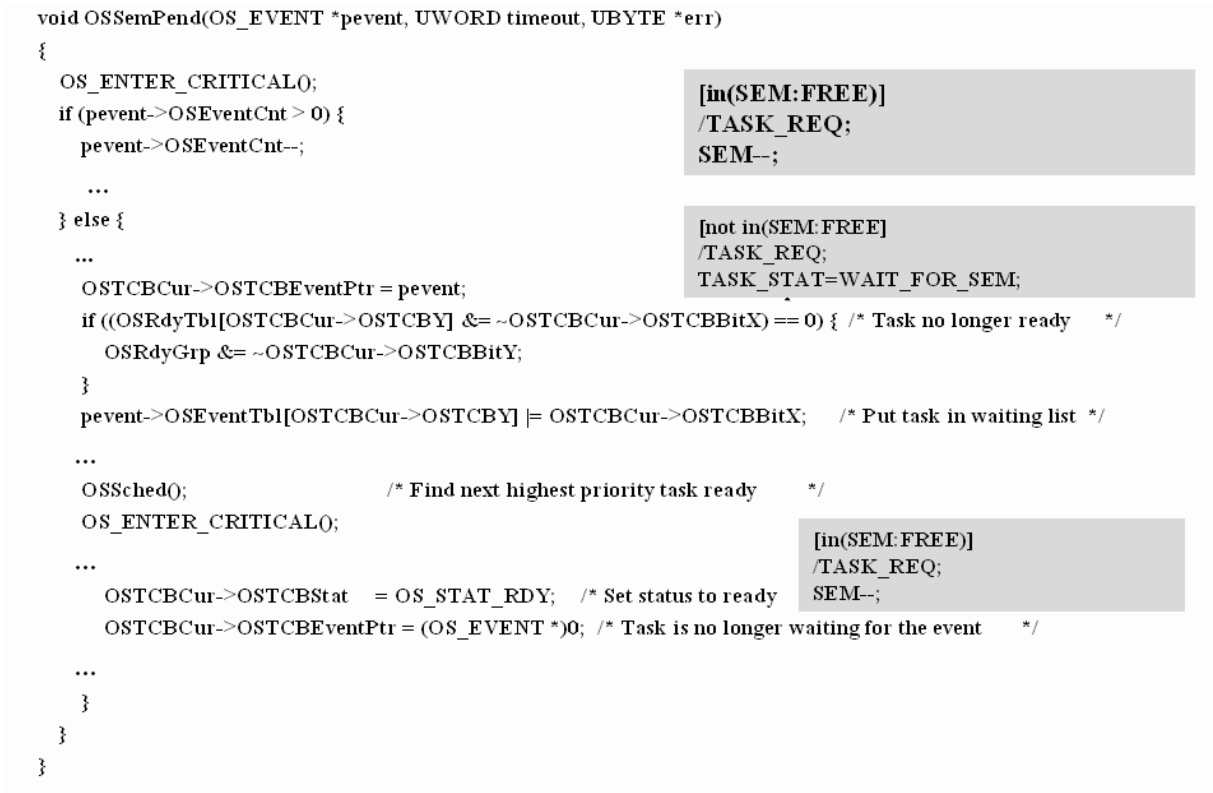
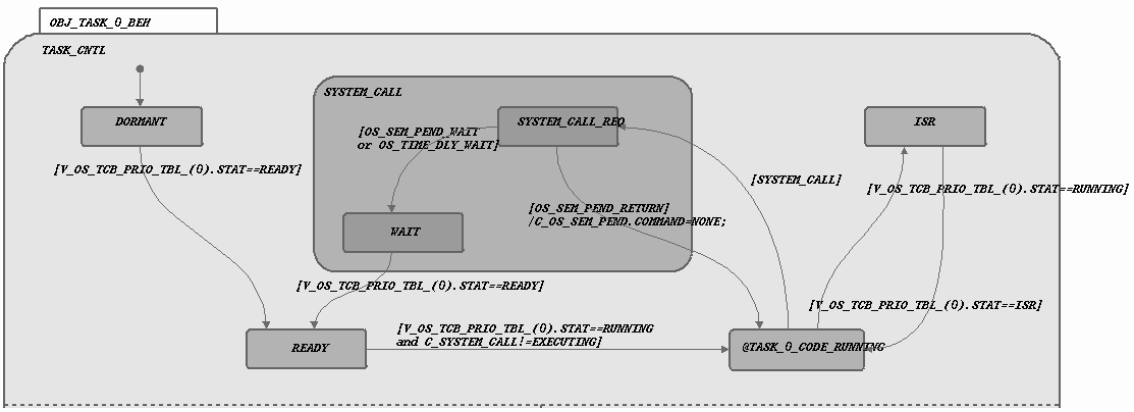
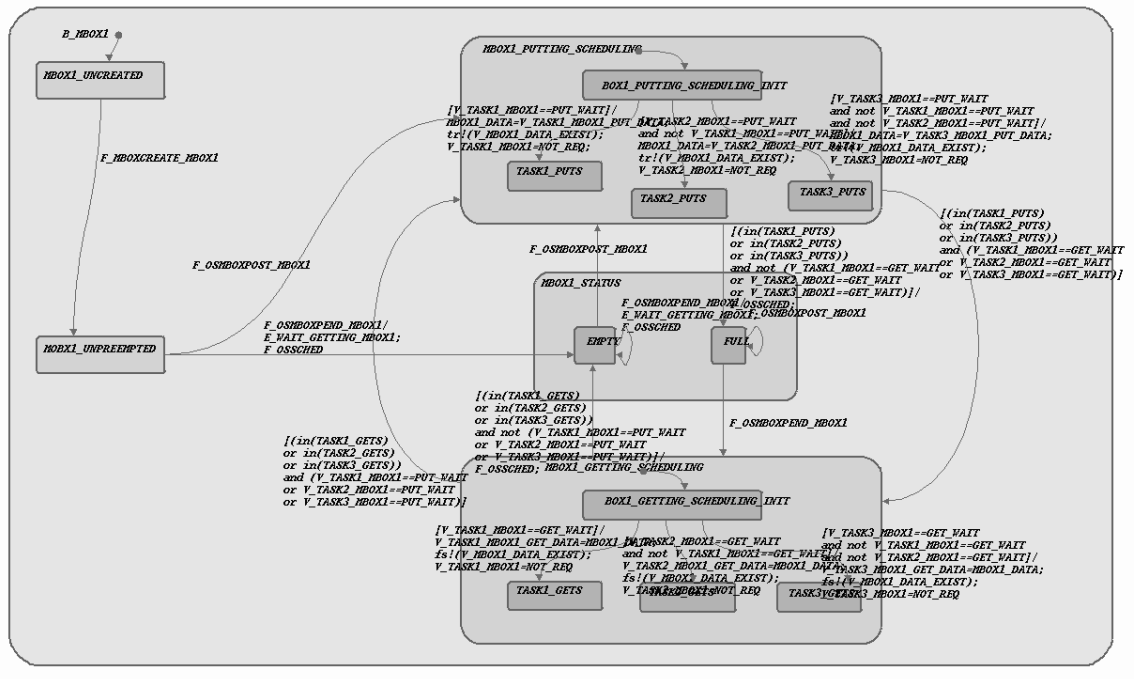


Figure 7: Implementation of semaphore pending function

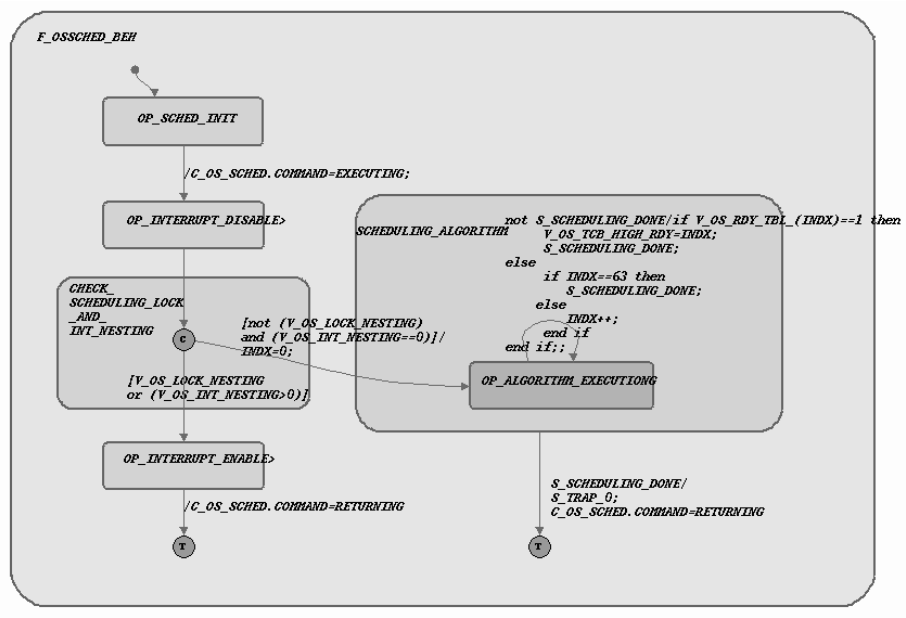


(a) Task resource

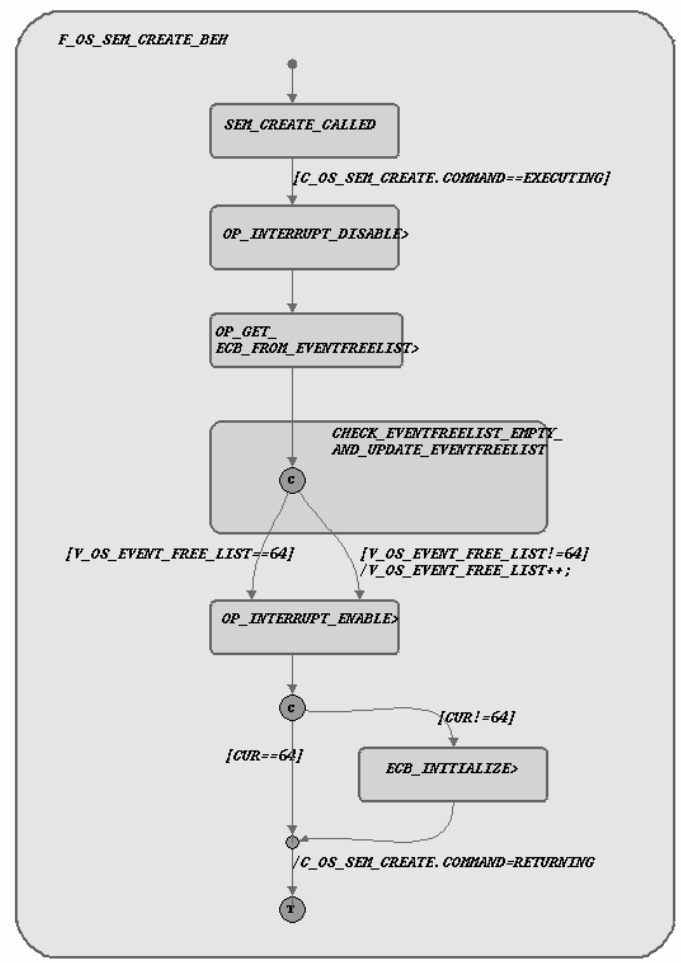


(b) Message mailbox resource

Figure 8: The resource models



(a) Scheduling function



(b) Semaphore creating function

Figure 9: The function models