Formal Modeling and Verification of Operational Flight Program in a Small-Scale Unmanned Helicopter

Dong-Ah Lee[1]; Sangkyung Sung[2]; Junbeom Yoo[3]; and Doo-Hyun Kim[4]

Abstract: Formal verification has played an important role in demonstrating correctness of safety-critical systems. Small-scale unmanned helicopters have been increasingly developed for various purposes such as scientific exploration and commercial or defense applications. The HELISCOPE project in Korea aims to develop a small-scale unmanned helicopter and its onboard embedded computing system for flight control and real-time transmission of multimedia data. This paper shares the authors' experience on the formal verification of the operational flight program (OFP) developed in the project. Because the OFP provides real-time controls on various sensors and actuators, demonstration of its correctness through formal verifications has been strongly recommended. This paper focuses on real-time process communications among sensing processes, a monitoring process, and a controller process. They all share a common data area. Two different formal models were developed for the OFP and verified with the SPIN and UPPAAL model checkers. While the SPIN model checker is widely used for modeling and verifying communication protocols, the real-time behavior of the OFP controller needs more advanced techniques that can handle real-time properties explicitly, i.e., timed automata and the UPPAAL verification system. The verification of the OFP found several safety-critical faults; they were all reported to development teams and fixed. **DOI: 10.1061/(ASCE)AS.1943-5525.0000165.**

CE Database subject headings: Verification; Models; Communication; Aircraft.

Author keywords: Formal verification; Formal modeling; Process communication; Operational flight program; SPIN; UPPAAL.

Introduction

The HELISCOPE project (Kim et al. 2009a) aims to develop an onboard embedded computing system and application services for an unmanned helicopter. It will be used for responding to disasters such as forest fires or volcanic eruption. The operational flight program (OFP) is a control program that provides real-time controls over sensors and actuators that are installed in a helicopter. The OFP developed in the project includes six independent processes and one shared data area. Four sensing processes read data from the sensors and write it into a shared data area, while one monitoring process controls their mutual exclusiveness. One control process also reads data from the shared data area and makes control commands for servomotors in real time and periodically. Correctness of the OFP should be demonstrated sufficiently, since the six independently executed processes show complex real-time behaviors.

This paper demonstrates the correctness of the OFP, concerning real-time process communications, using formal verification techniques (Peled 2001). A former work (Lee 2010a) dealt with the communications between four sensing processes and one control process, while the monitor process monitors the four sensing processes. Their behavior and communications were modeled in a formal modeling language PROMELA (protocol meta language) and formal verification was performed using the model checker SPIN (Holzmann 1997). However, the controller in the OFP has a strict timing bound in executing periodically, and the SPIN could not handle the real time behaviors precisely. UPPAAL (Bengtsson et al. 1995) is an automatic verification system which uses timed automata (Alur and Dill 1994) as an input front end. The real-time behavior of the OFP was modeled efficiently with the timed automata and UPPAAL, and verified against important properties such as

1. The semaphores on four reading processes should function correctly;
2. All reading processes should access the shared data area safely;
3. Simultaneous reading and writing to the shared data area should be safe; and
4. The control process should be able to get required data within its timing bound.

Properties 1 and 2 can be verified with the SPIN model checker, while Properties 3 and 4 need the timed automata model and the UPPAAL verification system. The verification on the OFP found some important faults, and they all were reported to development teams. The latest version of the OFP has no such fault regarding real time process communications.

The paper is organized as follows. Background briefly explains the small-scale unmanned helicopter developed in the HELISCOPE project. It also explains formal verifications using the SPIN and UPPAAL model checkers, which are pertinent to the discussion. Formal Modeling and Verification Using SPIN describes the OFP model specified in the PROMELA programming language and the verification results using the SPIN model checker. Limitations met while using the SPIN are also described. To overcome the limitations, timed automata and the UPPAAL verification system as described in

[1]Ph.D. Student, College of Information and Communications, Konkuk Univ., Seoul 143-701, Korea. E-mail: ldalove@konkuk.ac.kr

[2]Associate Professor, Dept. of Aerospace Information Engineering, Konkuk Univ., Seoul 143-701, Korea. E-mail: sksung@konkuk.ac.kr

[3]Assistant Professor, College of Information and Communications, Konkuk Univ., 1 Hwayang-dong, Gwangjin-gu, Seoul 143-701, Korea (corresponding author). E-mail: jbyoo@konkuk.ac.kr

[4]Associate Professor, College of Information and Communications, Konkuk Univ., Seoul 143-701, Korea. E-mail: doohyun@konkuk.ac.kr

Note. This manuscript was submitted on January 24, 2011; approved on September 16, 2011; published online on September 19, 2011. Discussion period open until March 1, 2013; separate discussions must be submitted for individual papers. This paper is part of the ***Journal of Aerospace Engineering***, Vol. 25, No. 4, October 1, 2012. ©ASCE, ISSN 0893-1321/2012/4-530–540/$25.00.

Real-Time Verification using UPPAAL were used. Related Work introduces related work on formal verification in the aerospace industry. The paper is concluded in Conclusion and Future Work.

Background

Small-Scale Unmanned Helicopters

A helicopter makes a dynamic flight through lift force and thrust, both generated by fast rotating rotor blades and their angular deflections (Kim et al. 2009a). The deflection angles of the main and tail rotor blades play an important role for high maneuverability. The main blade pitch is typically controlled by a swash plate connected to the helicopter flight control servos. The tail rotor is connected through a combination of a drive shaft and gearbox along the tail boom. Collective pitch, longitudinal cyclic, lateral cyclic, and the tail rotor collective can provide an independent control plan for altitude, forward/backward, left/right, and directional motion, respectively. The most prominent advantage of a helicopter system is that it can provide advanced maneuvering capabilities such as hovering or nose-in-circle flight, which is impossible for fixed wing aircrafts. A major drawback, however, comes from difficult control problems in maintaining a stable attitude, caused by its complicated nonlinear aerodynamic mechanism. Therefore, it needs a series of complicated control logics such as SAS (stability augmentation system) control, velocity hold, position hold, direction hold, altitude hold, and its combined algorithm. Control logics and sensor data processing are managed with an onboard computing system as well as communication and control from ground control system (GCS), despite its constraints on weight and volume. In this respect, a very efficient and powerful onboard computing system design and its verification are highly required. Fig. 1 shows a test flight of the helicopter developed in the HELISCOPE project.

A small-scale unmanned helicopter in the HELISCOPE project, which is pertinent to the discussion, receives commands from GCS (ground control system) in real time. OFP (operational flight program) running on FCC (flight control computer) installed in the helicopter makes control signals for SWM (helicopter servoactuator switching module). It also reads navigation information from GPS/INS (global positioning system/inertial navigation system) and AHRS (attitude and heading reference system). The OFP periodically sends the vehicle's information of current location and attitude to the GCS for monitoring purposes. For flight control purposes, the autopilot control logics within the OFP controller processes navigation data and sends control outputs to the SWM for servocontrol. Fig. 2 shows an overview of communications between all processes in the OFP, illustrated from the viewpoint of formal verification.

Fig. 1. Test flight of a small-scale unmanned helicopter

The OFP consists of six independent processes and one shared data area. Four sensing processes read data from sensors and write the data into the shared data area, while one process controls their serialization (mutual exclusion). In addition to those aforementioned, the OFP has one control process, reading data from the shared data area, calculating and sending control data to actuators. Details of the four sensing processes are as follows: Reader 00 reads packets containing AHRS information, while Reader 01 reads navigation information and GPS data from a GPS installed in the helicopter. Reader 02 collects information from GCS, and Reader 03 collects real-time operational information of the helicopter. These four processes write their information on the shared data area ["Object Data Store" (ODS) in Fig. 2]. The processes Reader 00 and Reader 01 share the data area ODS 00 to write on. They also access ODS 01 to read from. The process Reader 03 writes the information of the helicopter onto ODS 02, while the process monitor provides them with semaphore facilities. "Controller" in Fig. 2 is the main controller of the OFP. It uses the data from GCS or values that are calculated by the Autopilot module with all of the data from the sensors. The controller reads the data stored in the shared data area and controls servomotors periodically.

The OFP should demonstrate its correctness concerning process communications through the shared data area. The monitor process should provide four input processes with correct serialization, while the controller process should be able to read data within a predefined timing bound. Any conflicts should be avoided between the controller and sensing processes in real time. Freedom from deadlock is also one of the important features to be demonstrated thoroughly.

Formal Verification

Formal methods (Wing 1990) encompass formal specification and formal verification. Formal specification is a technique for specifying (modeling) a system on the basis of mathematics and logic. Various techniques and notations, e.g., algebra, logic, decision table, graphics, and automata can be used. After completing the formal specification, formal verification techniques can be applied to the specification to prove whether the formal model of a system satisfies required properties. There are two main approaches in formal verification: deductive reasoning and algorithmic verification.

Deductive reasoning uses axioms and proof rules to establish the reasoning (verification). Experts construct the proofs by hand, and it usually requires great expertise in mathematics and logic. Even if tools known as theorem provers, e.g., PVS (Owre et al. 1992), HOL (Gordon and Melham 1993), can provide a certain degree of automation, the reasoning procedure itself is too big an obstacle to be used widely. The algorithmic verification is called model checking (Clarke et al. 1986; Clarke et al. 1999; McMillan 1993; Huang and Cheng 1998). It has been widely used in industry (Havelund et al. 1998; Lee et al. 2010b; Yoo et al. 2009), because it verifies finite-state systems through searching all states' space exhaustively to check whether specified correctness conditions are satisfied or not. It performs the verification automatically, but is restricted to the verification of finite-state systems because of state explosion problems. Deductive reasoning, on the other hand, has no such limitations. Representative model checking tools include SMV (McMillan 1993), SPIN (Holzmann 1997), VIS (Brayton et al. 1996), and UPPAAL (Bengtsson et al. 1995). Code verification is also an active research topic, and tools

J. Aerosp. Eng. 2012.25:530-540.

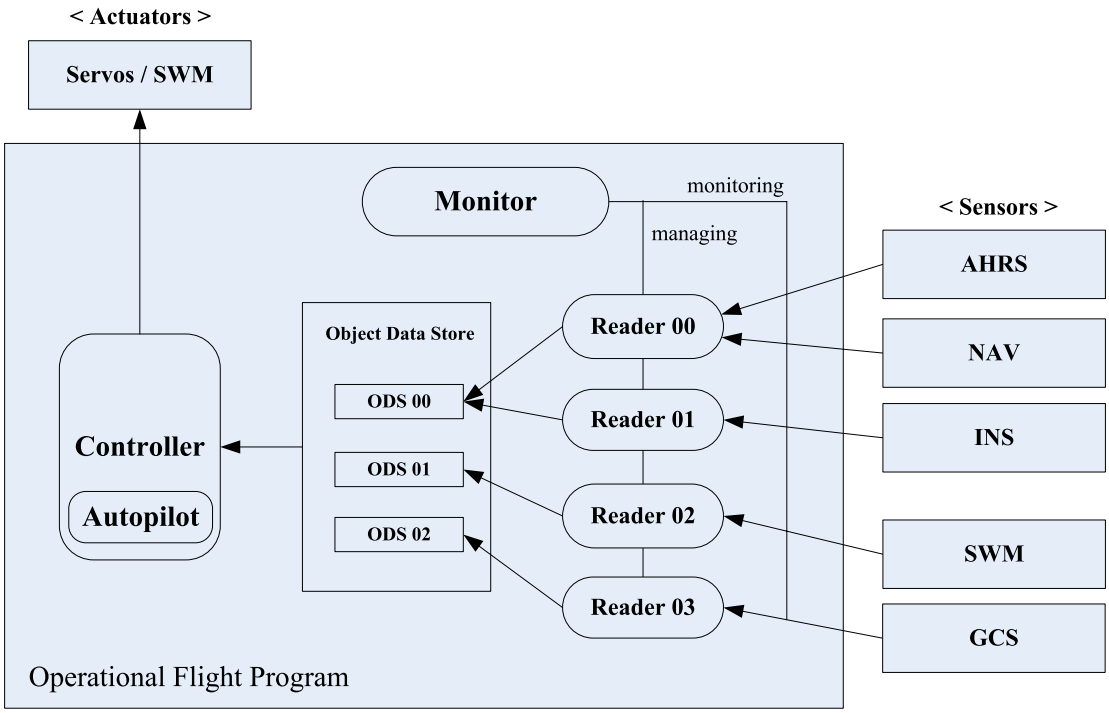

Fig. 2. An overview of process communications in the OFP

such as CBMC (Clarke et al. 2004) and BLAST (Henzinger et al. 2003) work directly on C programs.

With respect to the verification of communicating processes in the OFP, the latter, model checking, is more efficient and cost effective than the former, theorem proving. The main drawback of the former, requiring considerable expertise, makes the model checking techniques better suited for the verification of process communications. Indeed, as the performance of the model checking technique and the computation power of computers have increased rapidly, it has shown more improved performance and efficiency.

Model Checkers

SPIN (Holzmann 1997) is a widely used model checker for verifying process communications efficiently. If precise models of timing constraints are required, UPPAAL (Bengtsson et al. 1995) will be more useful. A general purpose model checker, Cadence SMV (McMillan 1993), can also be used. In the HELISCOPE project, SPIN was used for verifying the correctness of reading processes, which communicate with each other through the shared data area. UPPAAL was also used to model and verify real-time scheduling of the OFP controller. A brief introduction (Berard et al. 2001) to the SPIN and UPPAAL is as follows.

The model checker SPIN is a verification tool developed at Bell Laboratories, designed for simulation and verification of distributed software systems. The system under SPIN verification needs to be described in PROMELA (Holzmann et al. 1991), SPIN's formal specification language. PROMELA can describe not only the behavior of individual process but also interactions between processes. The SPIN model checker can be used efficiently in two ways: simulation and verification. The behavior of the system model can be simulated in steps: automatic or batch mode. SPIN also can verify specific properties written in PLTL (propositional linear temporal logic) of the system model. If the checking fails, it produces a counterexample, a scenario leading to the failure visually.

UPPAAL is an integrated environment for modeling, simulation, and verification of real-time systems specified in timed automata. It was developed jointly by Aalborg University and Uppsala University. UPPAAL provides powerful simulation particularly for real-time controllers and communication protocols, where real timing aspects are crucial. It can analyze networks of timed automata, communicating through channels or shared variables, with real valued clocks. UPPAAL consists of three main parts: description language, simulator, and model checker. The description language is a nondeterministic guarded command language. The simulator is a validation tool enabling examination of possible dynamic executions of a system. The last one, the model checker, can check invariant and reachability properties by exploring the entire state space of a system.

Formal Modeling and Verification using SPIN

This section describes how the OFP process communications were modeled with PROMELA and verified the model against important properties such as correct reading and mutual exclusion. Fig. 3 illustrates communications between four reading processes and their shared global data area. It also shows semaphore operations on the four reading processes provided by the monitor process. The in-line function was modeled to access (writing and reading) the shared memory area with a calling procedure *accessN()* in PROMELA as described in Fig. 4.

With the PROMELA model in Fig. 4, SPIN model checking was performed against Properties 1 and 2, which were introduced in the "Introduction" section. The properties are now refined into the following:

1. The semaphores managed by the monitor process on four reading processes should function correctly; and
2. Three processes, Reader 01, Reader 02, and Reader 03, should access the same global data mutually exclusively.

The simulation was first performed as described in Fig. 5 to confirm the correctness of the models. The calling procedure *accessN()* shows communications between the global shared data area and four reading processes in the simulation. This procedure works with mutex variables *mutex_A* ~ *mutex_E*; every process calls it to access the shared variables. Messages from the monitor

J. Aerosp. Eng. 2012.25:530-540.

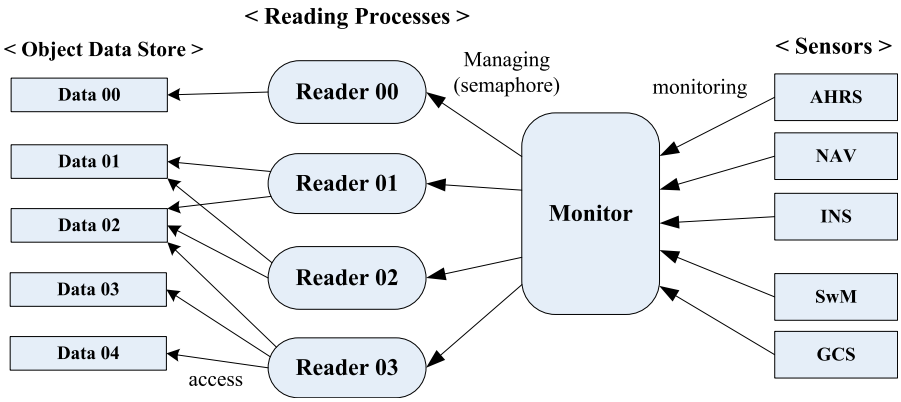

Fig. 3. A schema of the PROMELA model for the OFP's process communications

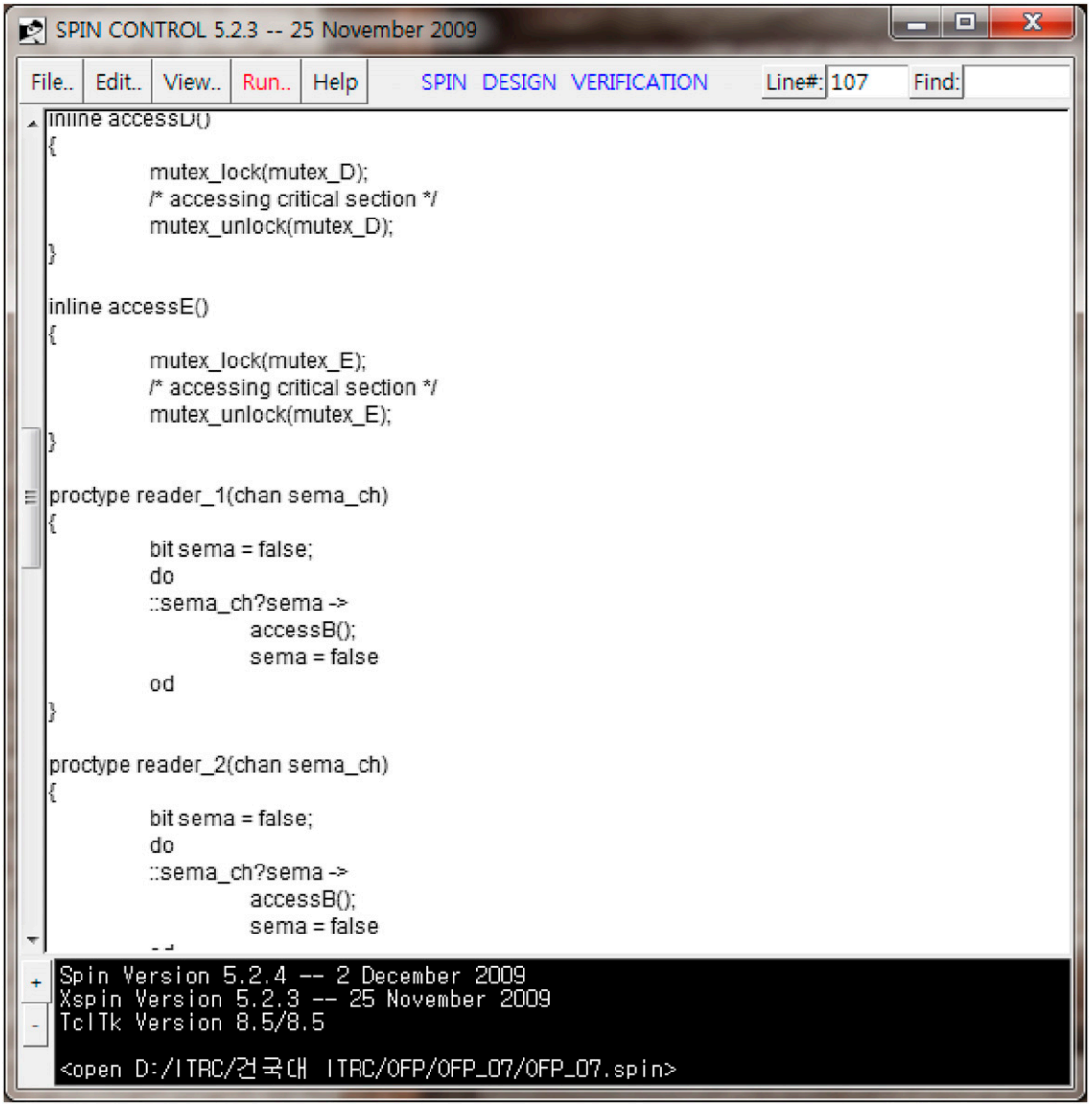

Fig. 4. A part of the PROMELA program

J. Aerosp. Eng. 2012.25:530-540.

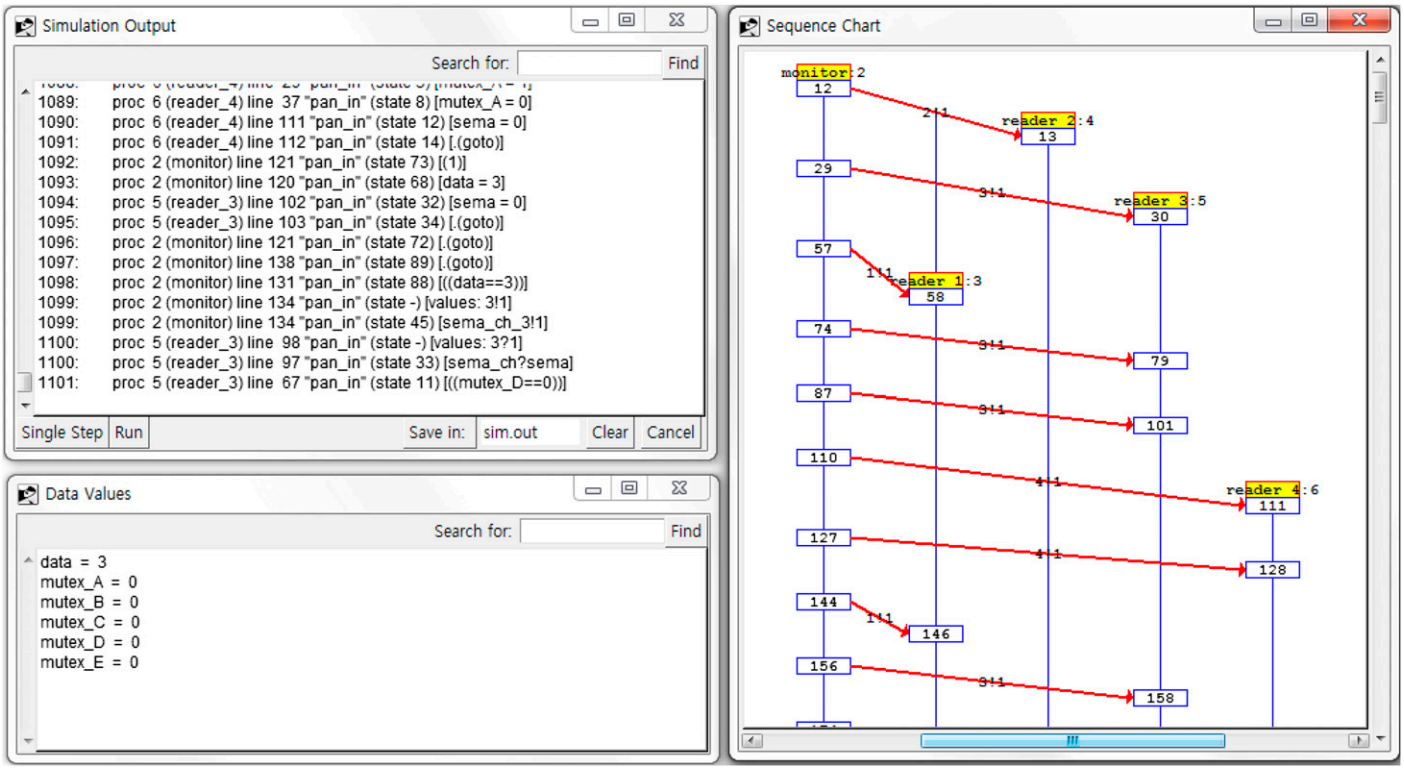

Fig. 5. A screen dump of the SPIN simulation

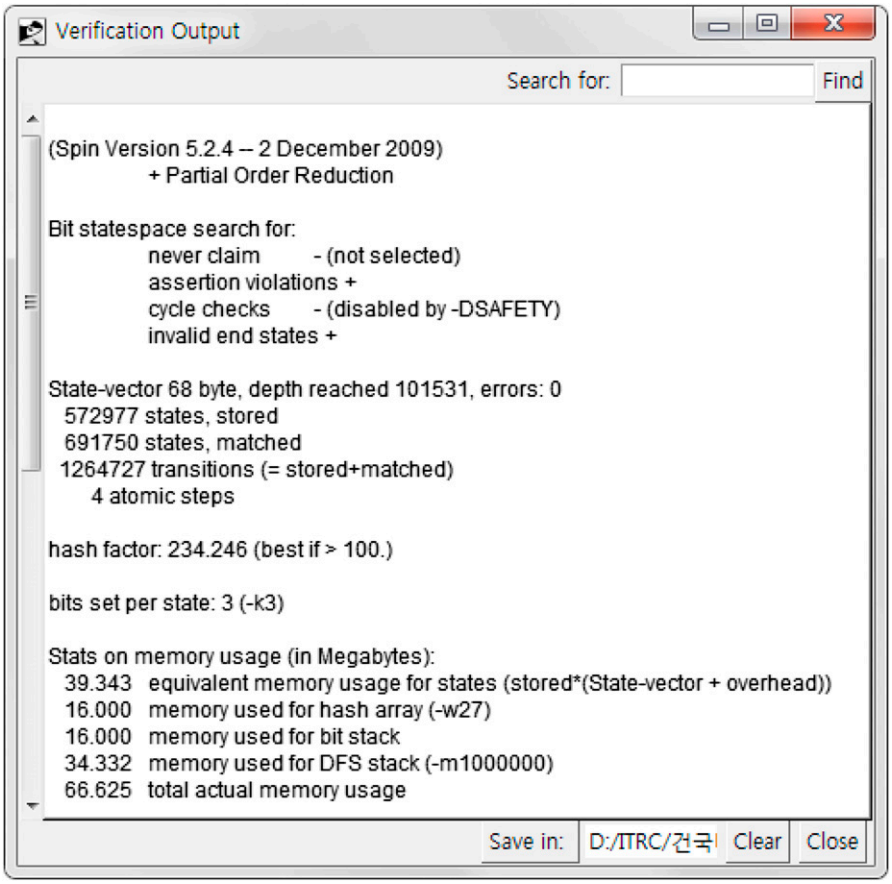

Fig. 6. A verification result by the SPIN model checker

J. Aerosp. Eng. 2012.25:530-540.

process in the simulation indicate the semaphore operations provided by the monitor process. After performing the simulation sufficiently, SPIN model checking was performed against the two properties, and SPIN proved that these properties were all satisfied, as shown in Fig. 6. The OFP provides semaphore operations correctly and there is no unsafe access to the shared data area by the four reading processes.

It is worth mentioning that the shared memory area in the OFP was modeled with calling procedures [i.e., *accessN()* procedure]. This was made possible by a strong advantage of SPIN, which is modeling communication protocols between independent processes through channels. However, it also has a limitation on modeling all the behaviors with other OFP parts such as the controller process. The controller process has strict timing scheduling and restrictions, and the behavioral difference between the real implementation (i.e., accessing the share data area in real time) and the PROMELA model (i.e., modeling it as procedure calling) might cause problems. To analyze the timing-related behavior of the controller precisely, it was decided to use the UPPAAL verification system and timed automata, as introduced in the following section.

Real-Time Modeling and Verification Using UPPAAL

Overview of the OFP

The OFP basically consists of TMO (time-triggered, message-triggered object) (Kim and Kopetz 1994; Kim 2009b) instances such as a one time-triggered task and four message-triggered tasks and ODS (object data store). It also contains periodic and nonperiodic sensors and actuators. The system was modeled with timed automata in the UPPAAL verification system to analyze its real-time behaviors more precisely. Fig. 7 depicts a schema of the timed automata model of the OFP. It has 16 processes consisting of four parts of the system: (1) a sensor part to generate and send data; (2) a monitor and reader part to receive data from sensors and store data in ODS; (3) an ODS part to be accessed by readers and controller; and (4) a controller part to read data stored in ODS.

Formal Modeling

Assumptions

This paper focuses on the real-time communications between OFP processes; not only the OFP processes but also the sensors need to be modeled. The time-triggered task, such as the controller process, runs with the timing (signal) generated by the timer process, while the message-triggered tasks, such as the reader and monitor processes, run with a message generated by external sensors. Modeling of the system is based on the assumptions as follows:

- Sensors generate and send a message exactly on their own periodic time;
- Data transition time between sensors and readers is ignored;
- Error messages are not considered; and
- The order of accessing shared data area, not the number of times, is considered.

Sensors

Four sensors named NAVsensor, GPSsensor, GCSsensor, and SWMsensor were modeled. Every sensor generates and sends messages via channels to the monitor process, which monitors the sensors and sends messages back to the reader processes. Every sensor except

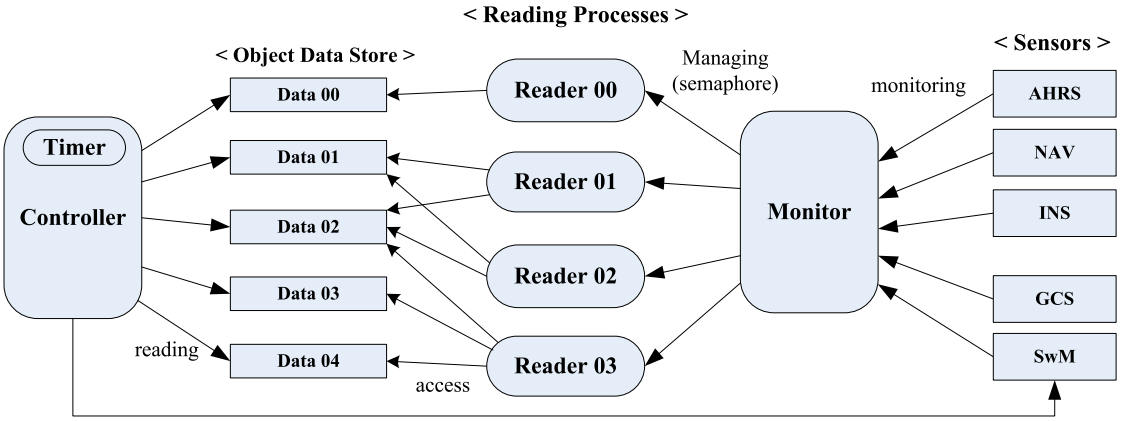

Fig. 7. A schema of timed automata model of the OFP

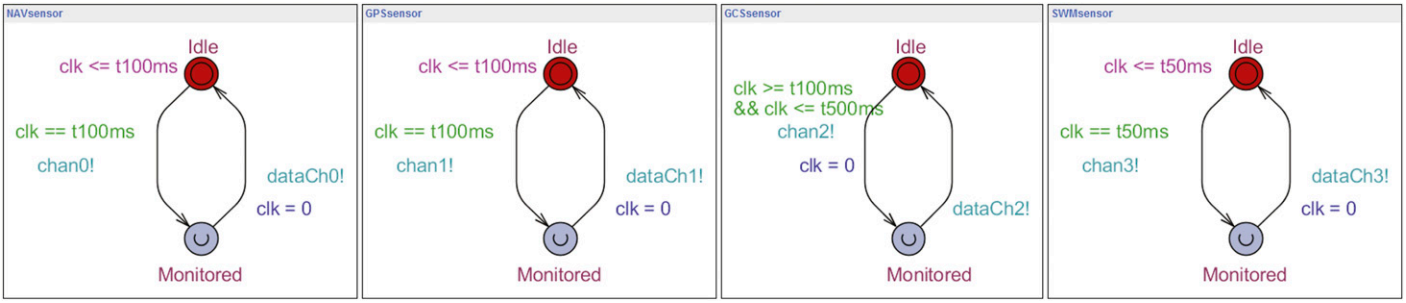

Fig. 8. Timed automata models for sensing processes

J. Aerosp. Eng. 2012.25:530-540.

GCSsensor has its own sensing cycle (timing) depending on the device. This model assumes that the processes send a message on a 10- or 20-Hz cycle. The GCSsensor process has no certain execution cycle, since it receives messages from the ground by command. The processes for the four sensors are illustrated in Fig. 8.

Monitor and Readers

Fig. 9 describes the monitor and reader parts with five processes. The monitor is a process that monitors four serial ports and manages four reader processes. It has four channels from the sensors and four channels to reader processes for providing semaphore facilities. If the Monitor process receives data from a sensor, it posts a semaphore of the corresponding process to an appropriate reading process. The other four processes are reader processes [Reader0, Reader1, Reader2, and Reader3] that wait for a corresponding semaphore until the monitor receives data and posts its semaphore. Each reader process accesses the shared data area mutually exclusively as verified in the previous section.

Object Data Store

Fig. 10 illustrates the object data store modeling with five processes. The OFP has five shared data areas, and all OFP processes use mutex variables and functions to access the shared data area. The mutex was modeled using channels. When a process requests for permission to access a critical section, it locks a mutex variable to prohibit accessing from other processes. Other processes have to wait for the unlocking of the mutex. A channel was assigned to each process to implement shared variables and operations such as *lock_mutex()* and *unlock_mutex()*. If a process requests preemption of a shared variable, then it sends a message through its channel. There is only one channel to change the state of SharedVar from Idle to Preempted. Therefore, other processes that request for access to SharedVar cannot send a message through the channel until SharedVar becomes Idle and preemption to the shared variable becomes possible.

Controller and Timer

The controller part consists of two processes: a timer (Ticker) and a controller (Controller) as shown in Fig. 11. The controller process accesses the shared data area while the reader processes also access

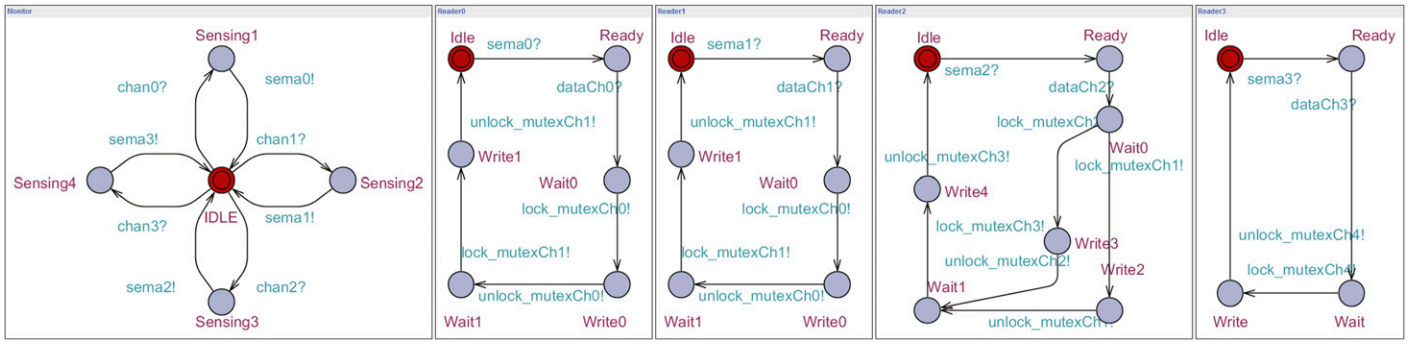

Fig. 9. Timed automata models for monitor and reader processes

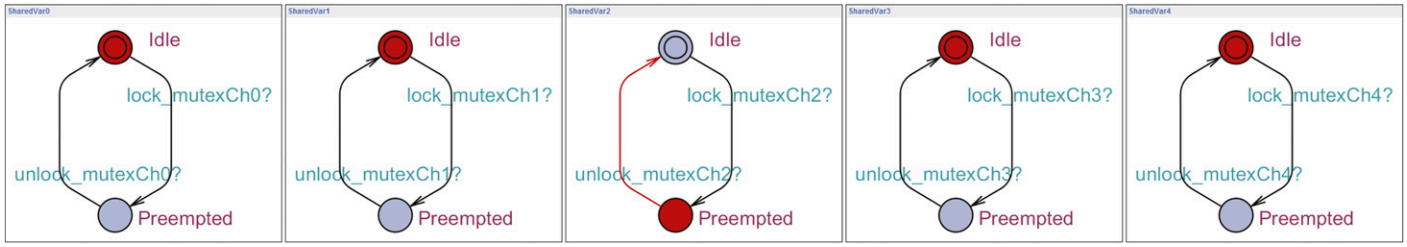

Fig. 10. Timed automata models for object data store

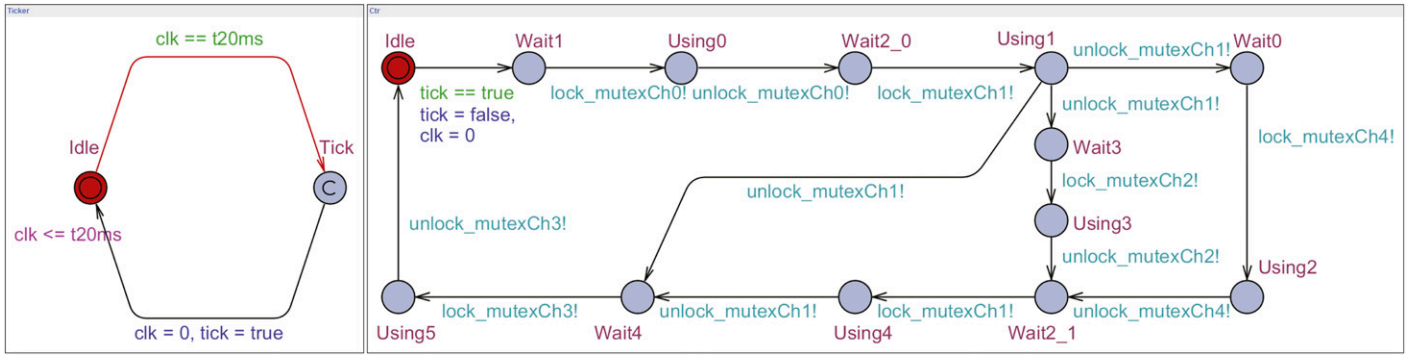

Fig. 11. Timed automata models for controller and timer processes

J. Aerosp. Eng. 2012.25:530-540.

Table 1. CTL properties for the UPPAAL Verification

Path formula	Property (P_N)	Description
AfP_0	*Ctr.Wait*0 && *Ctr.clk* > *t20ms* && *Reader*3.*Write*	Controller does not wait to access SharedVar0 over 20 ms while Reader3 accesses SharedVar0
AfP_1	*Ctr.Wait*1 && *Ctr.clk* > *t20ms* && (*Reader*0.*Write*1 || *Reader*1.*Write*1)	Controller does not wait to access SharedVar1 over 20 ms while Reader0 or Reader1 access SharedVar2
AfP_2	(*Ctr.Wait*2_0 || *Ctr.Wait*2_1) && *Ctr.clk* > *t20ms* && (*Reader*0.*Write*1 || *Reader*1.*Write*1 || *Reader*2.*Write*2)	Controller does not wait to access SharedVar2 over 20 ms while Reader0, Reader1, or Reader2 access SharedVar2
AfP_3	*Ctr.Wait*3 && *Ctr.clk* > *t20ms* && *Reader*2.*Write*3	Controller does not wait to access SharedVar3 over 20 ms while Reader2 accesses SharedVar3
AfP_4	*Ctr.Wait*4 && *Ctr.clk* > *t20ms* && *Reader*2.*Write*4	Controller does not wait to access SharedVar4 over 20 ms while Reader2 accesses SharedVar4

it simultaneously. The controller process calculates the next values of the control points (for servomotors) with the shared data and sends control values to the servos. The controller requires to run with a safe execution cycle and deadline, since too rapid or too slow control of the OFP often results in the UAV (unmanned aerial vehicle) being out of control. It was modeled in such a way that the controller process runs by a periodic message from the ticker process running with a 50-Hz period. This cycle is a deadline for the controller to access the shared data area and calculate the next control values also.

While the monitor process and all reader processes are message-triggered tasks, the controller is a time-triggered process. The TMO scheme regulates so that message-triggered methods (SvM: service method) cannot disturb the executions of time-triggered methods (SpM: spontaneous method). Therefore, it was defined that the controller has a higher priority than others such as the monitor and reader processes.

Verification Result

With the preceding timed automata model, the UPPAAL model checking was performed against Properties 3 and 4 introduced in the "Introduction" section. The properties are refined as follows:

- Each reader process (NavReader, GpsReader, AdtReader and SwmReader) should read data without losing; and

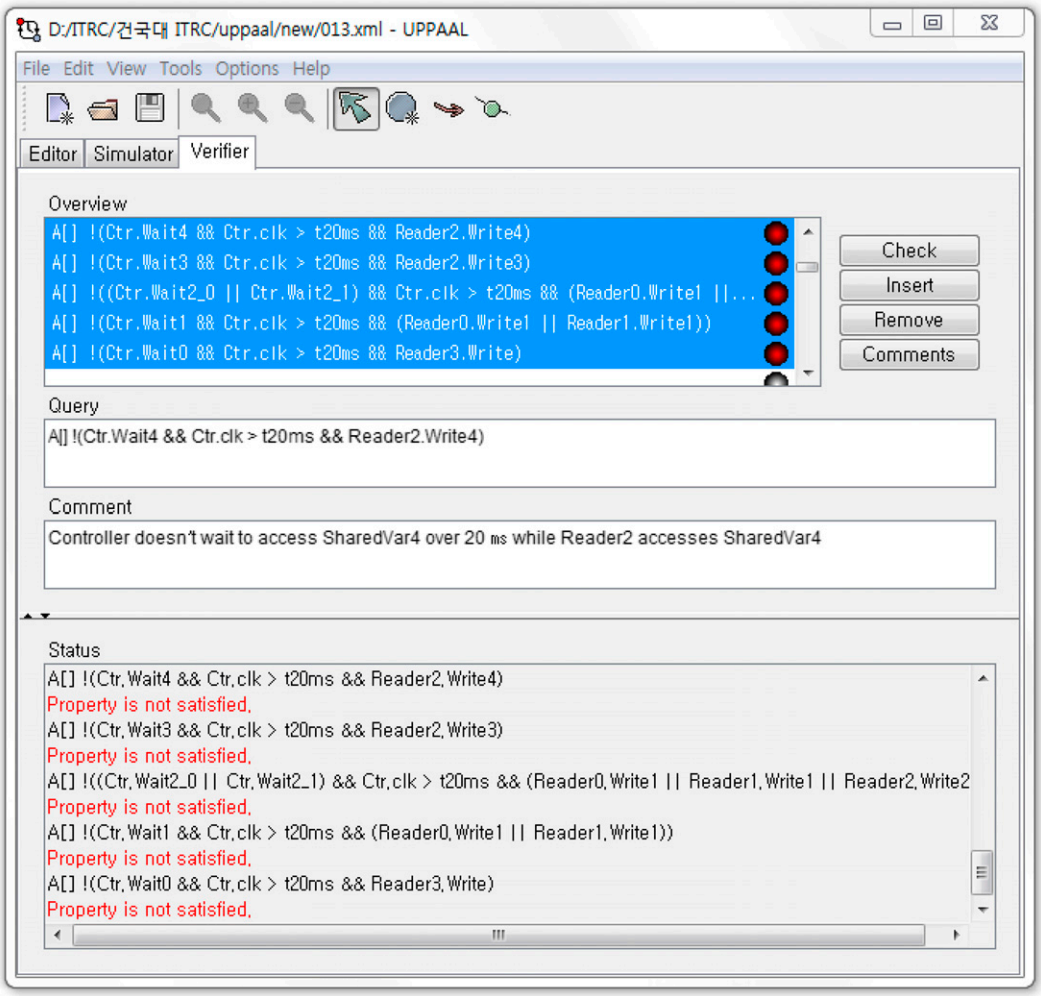

Fig. 12. Verification results by the UPPAAL verification system

J. Aerosp. Eng. 2012.25:530-540.

- The controller process should not wait to access SharedVar over its timing bound.

The controller process calculates output values for control data using the data from the five shared variables, and sends control values back to the SWM to control the helicopter. This process is the only one which has a deadline in this system. It should be strictly maintained that the controller runs within its deadline. The control process also should be able to access the SharedVar shared data area within its periodic execution time to use appropriate information for the current execution cycle.

The safety properties were defined to verify whether unexpected/incorrect situations occur. The safety property, which is expressed by the path formula $Af\varphi$ in UPPAAL, asks if the property φ is always true in all reachable states. If there is at least one state that satisfies the property φ, then UPPAAL results in *Not Satisfied* and provides a counterexample, which is a set of execution states to the failing condition of the property. It helps users analyze an invalid state of the system.

Table 1 shows the properties that were developed to verify the timed automata model for the OFP. Each property checks that the controller process does not wait to access the SharedVar due to the preemption of other reader processes. If the controller is not blocked over its deadline, then the UPPAAL verification system results in *Satisfied* for the property. On the other hand, if the controller is blocked beyond its time bound by other reader processes, then it results in *Not Satisfied*. It means the helicopter may lose its control during the moments the controller process is blocked.

The first property, AfP_0 means the controller should not wait to access the SharedVar0 over its deadline, since the Reader3 may write data on the shared data variable simultaneously. Meanings of the other properties can be interpreted in a similar way. For example, other properties verify whether the controller can access each SharedVar in a specific time. They are required to be distinguished for each other, since a running cycle or the access order of all Reader processes are different from each other. The properties may also be changed by the type of equipped sensors.

Fig. 12 shows the verification result of the CTL properties described in Table 1. All properties resulted in *Not Satisfied* (red lights). First, the AfP_0 property was checked again, and UPPAAL's diagnostic trace option was changed to simulate the counterexample. Fig. 13 shows the counterexample of the first property. A path could be traced to an invalid state with the simulation control on the left side of the UPPAAL window, and all variables and clocks for each step from an initial state to the invalid state could also be seen. The authors reported those results to the OFP developers and analyzed all counterexamples with them.

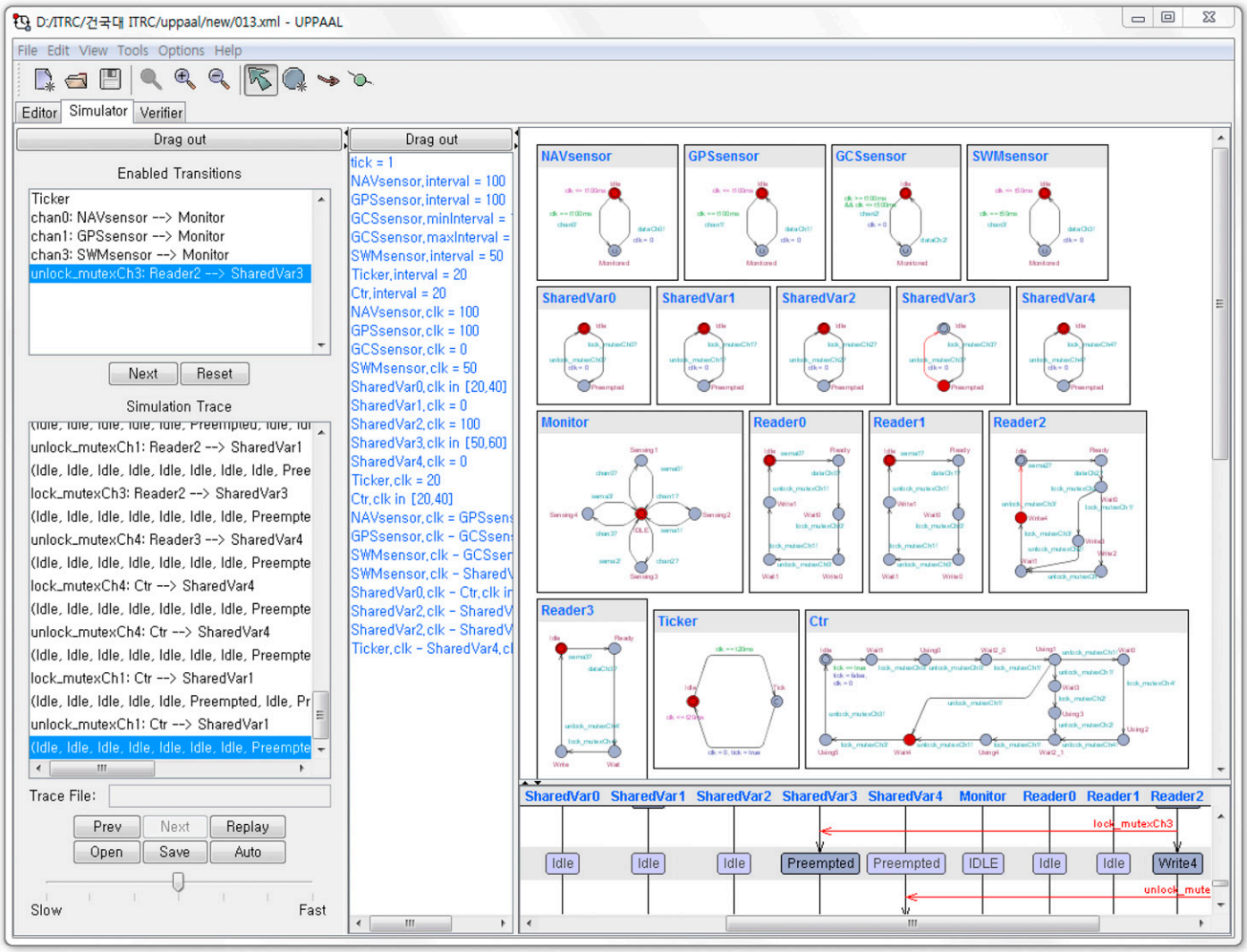

Fig. 13. A screen dump of a counterexample simulated by the UPPAAL simulator

J. Aerosp. Eng. 2012.25:530-540.

After cooperation with developers and verifiers, the causes that made the system go into the invalid states were found. One of these causes is that Reader0 should be ready for receiving data from the SWMsensor every 50 ms. If Reader0 does not have any restrictions, then it can preempt the SharedVar0 for 50 ms which is the cycle time of the SWMSensor. The Controller is supposed to run in 50 Hz (20 ms). Therefore, the reason for the *Not Satisfied* is that Reader0 can preempt SharedVar0 longer than the time for which the controller runs. It was also found that the problem might occur in not only AfP_0 but also $AfP_1 \sim AfP_4$. It was concluded that it was due to the fact that there is no time limitation to access the SharedVar. Two solutions were devised for the problem. The first is that the SharedVar has a time limitation to be accessed by all processes. The other is that every process which accesses the SharedVar has a time limitation to access the SharedVar for every access.

The model was modified and checked whether the Controller does not wait to access the SharedVar shared data area with the solution proposed. The first solution was implemented. A time limitation was added to the ODS template and the UPPAAL verification was performed again. The time limitation was added on the preempted state, so that processes can preempt the SharedVar only within the time limitation. A variable length of time limitation is also applied to the state to find the minimum length. Fig. 14 represents the modified ODS template with the minimum time limitation on the preempted state that was found, and Fig. 15 shows the verification result with the modified model. It can be seen that all properties result in *Satisfied* (green lights).

Related Work

The aerospace industry often uses formal verification techniques to verify nonfunctional properties such as reliability, security, and safety. In particular, research on formal verification of UAVs has been in progress. (Wu et al. 2009) used the model checker UPPAAL to verify some properties of UAV systems. They verified the UAV's flight control system, which is built on TTCAN (time-triggered controller area network), against important properties concerning reliability, security, scheduling, and fault-tolerance ability. They designed the system in two models, and developed 58 timed automata to verify them using UPPAAL. The approach applied the UPPAAL formal verification during the designing of the system, while the present study applies them after finishing the first phase of implementation. (Chaudemar et al. 2010) introduced safety architectures of autonomous systems. They applied the Event-B formal method, which supports the rigorous design of layered systems. The main properties they verified are about crash probability of and coordination of the activities between the layers. There is a study that used two different formal verification techniques, SAL (symbolic analysis laboratory) and UPPAAL, to model and verify UAV task allocation problems (Kasam 2008). It used these verification techniques simultaneously to verify two important constraints, nontiming and timing constraints, and compared results of the two techniques. The authors of this paper, on the other hand, applied two different techniques for different properties of the OFP.

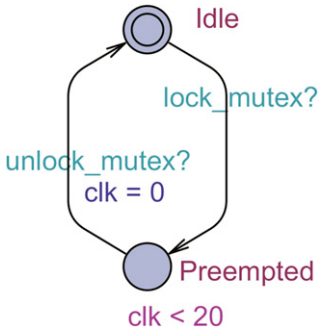

Fig. 14. Modified timed automata template for object data store

Conclusion and Future Work

This paper developed two different formal models of the OFP developed in the HELISCOPE project in Korea, and verified them against important properties concerning real-time process communications, using the SPIN model checker and the UPPAAL verification system. The SPIN model checker was used for verifying the correctness of communications between four reading processes and one shared data area. The model checker showed that all processes access the shared data area mutually exclusively.

Timed automata and the UPPAAL verification system were also used to verify the real-time behavior of the OFP. The verification found several possible faults that might cause the OFP controller to be stuck over its execution limits, i.e., it does not meet its timing constraint. Analysis on the verification results through simulations with counterexamples showed that it may occur when the controller waits to access the shared data area while one of the other processes accesses it. It may result in critical situations such as crashing into obstacles. The possible faults that were found were reported to the development teams and they analyzed the problem together with the authors, who suggested solutions for the problem, and applied one solution to the timed automata model and performed the formal verification again. It showed that the modified model satisfies the properties, and the solution then was used to solve the problem in the implementation of the OFP.

The authors are now planning to apply dynamic testing techniques to the OFP, which is real-time embedded software, and will compare the verification result from the model-level formal verifications with that from the code-level testing. There is also a plan to model the OFP more thoroughly. For example, a formal model of the TMO framework on which the OFP runs will help analyzing the OFP's real-time scheduling more precisely and thoroughly.

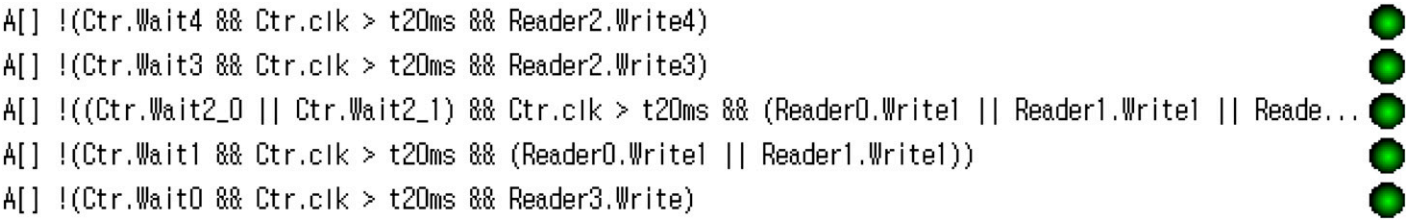

Fig. 15. Verification result of the modified model

J. Aerosp. Eng. 2012.25:530-540.

Acknowledgments

This research was supported by the MKE (The Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) support program supervised by the NIPA (National IT Industry Promotion Agency) (NIPA-2011-C1090-1131-0003) and (NIPA-2011-C1090-1131-0008).

References

Alur, R., and Dill, D. L. (1994). "A theory of timed automata." *Theor. Comput. Sci.*, 126(2), 183–235.

Bengtsson, J., Larsen, K. G., Larsson, F., Pettersson, P., and Yi, W. (1995). "UPPAAL—A tool suite for automatic verification of real-time systems." *Hybrid systems,* Springer, Heidelberg, Germany, 232–243.

Berard, B., et al. (2001). *Systems and software verification: Model-checking techniques and tools*, Springer, Heidelberg, Germany.

Brayton, R. K., et al. (1996). "VIS: A system for verification and synthesis." *Computer aided verification*, R. Alur and T. A. Henzinger, eds., Springer, Heidelberg, Germany, 428–432.

Chaudemar, J.-C., Bensana, E., and Seguin, C. (2010). "Model based safety analysis for an Unmanned Aerial System." *Proc., Dependable Robots in Human Environments 2010,* IEEE Robotics and Automation Society, Toulouse, France.

Clarke, E., Grumberg, O., and Peled, D. (1999). *Model checking*, MIT Press, Cambridge, MA.

Clarke, E. M., Emerson, E. A., and Sistla, A. P. (1986). "Automatic verification of finite-state concurrent systems using temporal logic specifications." *ACM Trans. Program. Lang. Syst.*, 8(2), 244–263.

Clarke, E. M., Kroening, D., and Lerda, F. (2004). "A tool for checking ANSI-C programs." *TACAS 2004*, K. Jensen, A. Podelski, eds., Springer, Heidelberg, Germany, 168–176.

Gordon, M. J. C., and Melham, T. (1993). *Introduction to HOL: A theorem proving environment for higher order logic*, Cambridge University Press, Cambridge, U.K.

Havelund, K., Lowry, M., and Penix, J. (1998). "Formal analysis of a space craft controller using SPIN." *Proc., 4th SPIN Workshop*, 749–765.

Henzinger, T. A., Jhala, R., Majumdar, R., and Sutre, G. (2003). "Software verification with BLAST." *Proc., 10th Int. SPIN Workshop*, Vol. 2648, Springer, 235–239.

Holzmann, G. J., (1991). *Design and validation of computer protocols*, Prentice Hall, Englewood Cliffs, NJ.

Holzmann, G. J. (1997). "The model checker spin." *IEEE Trans. Software Eng.*, 23(5), 279–295.

Huang, S., and Cheng, K. (1998). *Formal equivalence checking and design debugging. Frontiers in electronic testing*, Kluwer Academic, Boston.

Kasam, S. (2008). "Formal verification of time constrained UAV task allocation using model-checking. MS thesis, Univ. of Cincinnati, Cincinnati, OH.

Kim, D.-H., Nodir, K., Chang, C.-H., and Kim, J.-G. (2009a). "HELISCOPE project: Research goal and survey on related technologies." *Proc., IEEE 12th Int. Symp. on Object/Component/Service-Oriented Real-Time Distributed Computing in Tokyo*, IEEE Computer Society Press, Los Alamitos, CA, 112–118.

Kim, J. G., et al. (2009b). "TMO-eCos2.0 and its development environment for timeliness guaranteed computing." *Proc., 2009 Software Technologies for Future Dependable Distributed Systems*, IEEE Computer Society, Washington, DC, 164–168.

Kim, K., and Kopetz, H. (1994). "A real-time object model RTO.k and an experimental investigation of its potentials." *Proc., 18th Int. Annual Computer Software and Applications Conf., 1994 (COMPSAC 94)*, IEEE, Washington, DC, 392–402.

Kim, S.-G., Song, S.-H., Chang, C.-H., Kim, D.-H., Heu, S., and Kim, J.-G. (2009c). "Design and implementation of an operational flight program for an unmanned helicopter FCC based on the TMO scheme." *Proc., SEUS 2009*, Int. Federation for Information Processing, Newport Beach, CA, 1–11.

Lee, D.-A., Yoo, J., and Kim, D. (2010a). "Formal verification of process communications in operational flight program for a small-scale unmanned helicopter." *Proc., 6th Int. Conf. on Intelligent Unmanned Systems (ICIUS)*, Bali, Indonesia, 91–96.

Lee, D.-A., Yoon, S., Lee, M.-Y., Jin, H.-W., and Yoo, J. (2010b). "Formal verification of protocol stack for most network service using spin." *Proc., Korea Computer Congress*, Korean Institute of Information Scientists and Engineers, Seoul, 60–61.

McMillan, K. (1993). *Symbolic model checking*, Kluwer Academic, Boston.

Owre, S., Rushby, J. M., and Shankar, N. (1992). "PVS: A prototype verification system." *CADE 1992*, D. Kapur, ed., 748–752.

Peled, D. (2001). *Software reliability methods. Texts in computer science*, Springer, New York.

Wing, J. M. (1990). "A specifier's introduction to formal methods." *Computer*, 23(9), 8–22.

Wu, X., Ling, H., and Dong, Y. (2009). "On modeling and verifying of application protocols of TTCAN in flight-control system with UPPAAL." *Proc., 2009 Int. Conf. on Embedded Software and Systems*, IEEE Computer Society, Washington, DC, 572–577.

Yoo, J., Jee, E., and Cha, S. D. (2009). "Formal modeling and verification of safety-critical software." *IEEE Software*, 26(3), 42–49.