Experimental Evaluation of Unmanned Aerial Vehicle System Software Based on the TMO Model

Hansol Park and Doo-Hyun Kim
Department of Computer Science, Konkuk University, Seoul, Korea
{parkhs, doohyun}@konkuk.ac.kr

Jung-Guk Kim
Hankuk University of Foreign Stuidies, Korea
jgkim@hufs.ac.kr

Chun-Hyon Chang
Department of Computer Science, Konkuk University, Seoul, Korea
chchang@konkuk.ac.kr

Received 11 August 2008; Accepted 10 November 2008

Over the past few decades, a considerable number of studies have been conducted on the technologies to build an UAV (Unmanned Aerial Vehicle) control system. Today, focus in research has moved from a standalone control system towards a network-centric control system for multiple UAV systems. Enabling the design of such complex systems in easily understandable forms that are amenable to rigorous analysis is a highly desirable goal. In this paper, we discuss our experimental evaluation of the Time-triggered Message-triggered Object (TMO) structuring scheme in the design of the UAV control system. The TMO scheme enables high-level structuring together with design-time guaranteeing of accurate timings of various critical control actions with significantly smaller efforts than those required when using lower-level structuring schemes based on direct programming of threads, UDP invocations, etc. Our system was validated by use of environment simulator developed based on an open source flight simulator named FlightGear. The TMO-structured UAV control software running on a small computing platform was easily connected to a simulator of the surroundings of the control system, i.e., the rest of the UAV and the flight environment. Positive experiences in both the TMO-structured design and the validation are discussed along with potentials for future expansion in this paper.

Categories and Subject Descriptors: System & Architecture [**Real-Time System**]
General Terms: Real-Time Software Design and Simulation
Additional Key Words and Phrases: Unmanned Aerial Vehicle, Embedded System

---

1. INTRODUCTION

Over the past few decades, a considerable number of studies have been conducted on the technologies to build an UAV (Unmanned Aerial Vehicle) control system [Park et al. 2007]. Today, focus in research has moved from a standalone control system towards a network-centric control system for multiple UAV systems. Enabling the design of such complex systems in easily understandable forms that are amenable to rigorous analysis is a highly desirable goal. In this paper, we discuss our experimental evaluation of the Time-triggered Message-triggered Object (TMO) structuring scheme in the design of the UAV control system. The TMO scheme enables high-level structuring together with design-time guaranteeing of accurate timings of various critical control actions with significantly smaller efforts than those required when using lower-level structuring schemes based on direct programming of threads, UDP invocations, etc. Our system was validated by use of environment simulator developed based on an open source flight simulator named FlightGear. Positive experiences in both the TMO-structured design and the validation are discussed along with potentials for future expansion in this paper [Horowitz et al. 2003; Doherty 2004; Ippolito 2005].

The TMO (Time-triggered Message-triggered Object) Model formalized earlier by Kim and his collaborators [Kim 1997] has been found to be a sound real-time object model that can be used for various types of hard and soft real-time distributed computing applications. With TMO model, both function and timing behaviors of a system in distributed computing environment can be specified in explicit and natural easy-to-understand forms. To support execution of TMO's, several engines have been developed in the form of middleware layered on a few widely used OS platforms. Representative cases are TMOSM [Kim et al. 1999] on MS Windows XP, Windows CE, Linux and LTMOS [Kim et al. 2002] on Linux. In the work reported in this paper, we used TMOSM made by DREAM Laboratory in University of California, Irvine.

In this paper, we propose a TMO-based high-level design and implementation method for the real-time embedded software parts of the UAV control system. The proposed method covers a wide range of topics which include the development of a network-centric application for multiple autonomous UAV control systems. The main goal here is to improve the software engineering productivity and the software reliability to a significant extent. Improvements are sought for in various phases of engineering UAV control software such as design, implementation, and testing. As an experimental evaluation, our UAV control system software was validated by use of an environment simulator. FlightGear was used in a virtual flight environment, named Hardware-In-the-Loop (HIL) system, in which components can be replaced by actual hardware without rendering the remainder of the simulator inoperable [Sorton 2005]. The TMO-structured UAV control software running on a flight-qualified computing platform could be easily connected to a simulator of the surroundings of the control system, i.e., the rest of the UAV and the flight environment. The connection involved a TMO-structured bridge because some sensors and actuators in the UAV were not simulated in sufficient details and suitable forms in the FlightGear produced and maintained by a volunteer group.

In Section 2, as backgrounds, the basic structure of the TMO model, the type of UAV's considered, and the features of FlightGear are described briefly. In Section 3,

we present the design of an UAV control system based on the TMO model. In Section 4, our implementation and flight simulation are described. Finally, in Section 5, we conclude with a suggestion for future works.

2. BACKGROUNDS

2.1 TMO Structuring Scheme

We use the TMO model as a fundamental building-block and TMOSM (TMO Support Middleware) as the execution engine for our experiments [Kim 2000; Kim et al. 1999]. TMO is a natural, syntactically minor, and semantically powerful extension of the conventional object(s). Especially, TMO is a high-level real-time computing object. Member functions (i.e., methods) are executed within specified windows in the domain of global time. Such timing requirements are specified in natural intuitive forms with no esoteric styles imposed. As depicted in Figure 1, the basic TMO structure consists of four parts.

1) Spontaneous Method (SpM): A time-triggered (TT) method which is triggered when the real-time clock reaches specific values determined at design time and specified in AAC (Autonomous Activation Condition) as the time-windows for execution.
2) Service Method (SvM): A method similar to the conventional service method which is triggered by service request messages from clients.
3) Object Data Store (ODS): The set of data members which may be partitioned into ODS segments (ODSS's), each of which is a basic unit of storage that can be exclusively accessed by a certain TMO method at any given time or shared among executions concurrent of TMO methods (SpMs or SvMs).
4) Environment Access Capability (EAC): A list of entry points to remote object methods, logical communication channels, and I/O device interfaces. Major features are summarized below.
   (1) Distributed computing component: The TMO is a distributed computing component and thus TMO's distributed over multiple nodes may interact via remote method calls. To maximize the concurrency in execution of client methods in one node and server methods in the same node or different nodes, client methods are allowed to make non-blocking service requests to service methods. In addition, TMO's can interact by exchange of messages over RMMC's.
   (2) Clear separation between two types of methods: The TMO may contain two types of methods, time triggered (TT) methods (spontaneous methods or SpMs), which are clearly separated from the conventional service methods (SvMs). The SpM executions are triggered when the RT clock reaches time values determined at the design time. On the contrary, SvM executions are triggered by remote method calls from clients that are transmitted by the execution engine in the form of service request messages. Moreover, actions to be taken at real times, which can be determined at the design time, can appear only in SpMs. Triggering times for SpMs must be fully specified as constants during the design time. Those RT constants as well as related guaranteed completion times (GCTs) of the SpM appear in the first clause of an SpM

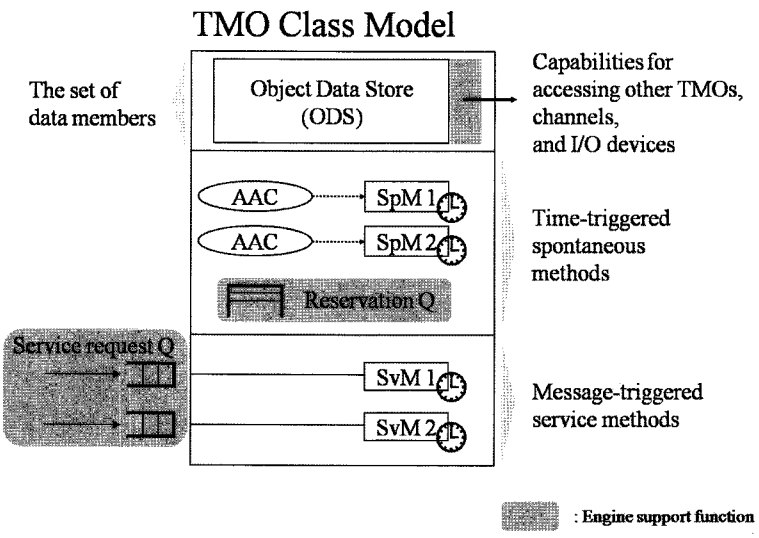


Figure 1. Structure of TMO class model.

specification called the autonomous activation condition (AAC) section. An example of an AAC is "for t=from 10am to 10:50am every 30min start-during (t, t+5 min) finish-by t+10 min" which has the same effect as {"start-during (10am, 10:05am) finish-by 10:10am", "start-during (10:30am, 10:35am) finish-by 10:40am"}.

(3) Basic concurrency constraint (BCC): This rule prevents potential conflicts between SpMs and SvMs and reduces the designer's efforts in guaranteeing timely service capabilities of TMOs. Basically, activation of an SvM triggered by a message from an external client is allowed only when potentially conflicting SpM executions are neither active nor likely to be disturbed. An SvM is allowed to execute only when an execution time-window big enough for the SvM exists and does not overlap with the execution time-window of any SpM that accesses the same ODSS's as the SvM. However, the BCC does not stand in the way of either concurrent SpM executions or concurrent SvM executions.

(4) Guaranteed completion time (GCT) of the server (i.e., an SvM of a server TMO) and the result return deadline imposed by the client: The TMO incorporates deadlines in the most general form. Basically, for output actions and method completions of a TMO, the designer guarantees and advertises execution time-windows bounded by start times and completion times. In addition, deadlines can be specified in the client's calls for service methods for the return of the service results.

2.2 TMOSM Architecture

The TMO Support Middleware (TMOSM) consists of a number of virtual machines (VMs), each managing a set of threads and using them to perform certain specialized functions as parts of executing TMO's (see Figure 2). To make VMs coexist on top of a commodity kernel, TMOSM contains one more component, which can be viewed

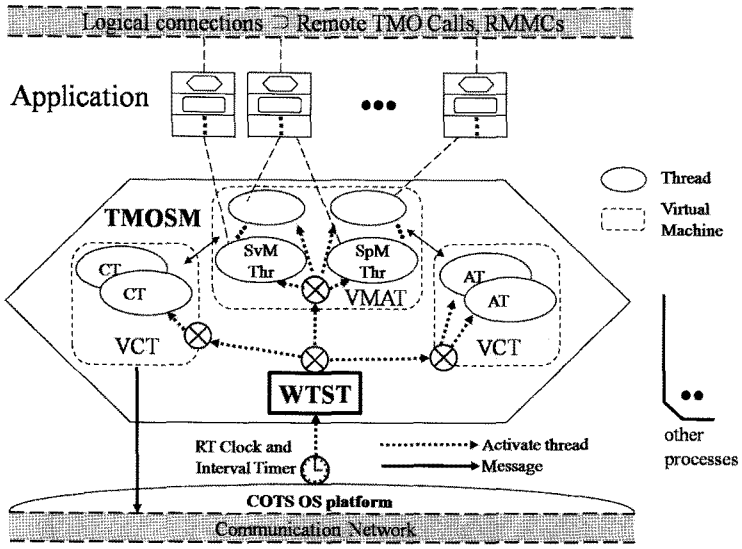


Figure 2. TMOSM architecture.

as the innermost core and is a super-micro thread called the WTST (Watchdog Timer & Scheduler Thread). It is a "super-thread" in that it runs at the highest possible priority level. It is also a "micro-thread" in that it manages the scheduling/activation of all VMs which in turn operate other threads in TMOSM. Even those threads created by the node OS before TMOSM starts are executed only if WTST allocates some time-slices to them. Therefore, WTST is in control of the processor and memory resources with the cooperation of the node OS kernel. WTST leases processor and memory resources to three VMs in a time-sliced and periodic manner. Each VM can be viewed conceptually as being periodically activated to run for a time-slice. Each VM is responsible for a major part of the functionality of TMOSM. Each VM maintains a number of application threads. In fact, whenever WTST assigns a time-slice to a VM, the VM in turn passes the time-slice onto one of the application threads that belong to it. The component in each VM that handles this "time-slice relay" is the application thread scheduler. For example, VM-A has the application-thread-scheduler, VM-A-Scheduler. The application thread scheduler is actually executed by WTST. To be more precise, at the beginning of each time-slice, a timer-interrupt results in WTST being awakened. WTST then determines which VM should get this new time-slice. If VM-A is chosen, WTST executes VM-A-Scheduler and as a result, an application thread belonging to VM-A runs for a time-slice as WTST enters into the event-waiting mode. The set of VMs is fixed at the TMOSM start time. One iteration of the execution of a specified set of VMs is called a TMOSM cycle. For example, one TMOSM cycle may be: VCT VMAT VAT VMAT. The following three VMs handle the core functions:

(1) VCT (VM for Communication Threads): The application threads maintained by this VM are those dedicated to handling the sending and receiving of middleware messages. Middleware messages are exchanged through the communication

network among the middleware instantiations running on different DC nodes to support interaction among TMO's. Therefore, these application threads are called communication threads and denoted as CTs in Figure 2. A communication thread also distributes middleware messages coming through the network to their destination threads, typically belonging to another VM discussed below.

(2) VMAT (VM for Main Application Threads): The application threads maintained by this VM are those dedicated to executing methods of TMOs with maximal exploitation of concurrency. Those application threads are called main application threads and denoted as MATs in Figure 2. Normally to each execution of a method of an application, TMO is dedicated to a main application thread. In principle, TMO method executions may proceed concurrently whenever there are no data conflicts among the method executions. Every time-slice not used by the other VMs is normally given to this VM. In every one of our prototype implementations of TMOSM, the application thread scheduler in VMAT uses a kind of a deadline-driven policy for choosing a main application thread to receive the next time-slice.

(3) VAT (VM for Auxiliary Threads): This VM maintains a pool of threads which are called auxiliary threads and denoted as ATs in Fig. 2. Some auxiliary threads are designed to be devoted to controlling certain peripherals under orders from TMO methods (executed by main application threads). Others wait for orders for executing certain application program-segments and such orders come from main application threads in execution of TMO methods. Use of this VAT has been motivated partly by the consideration that it should be easier to analyze the temporal predictability of the application computations handled by each VM, i.e., those handled by VMAT and those by VAT, than to analyze the temporal predictability of the application computations when there is no VAT and thus VMAT alone handles the combined set of application computations.

Also, WTST provides the services of checking for any deadline violations and if a violation is found, it provides an exception signal to the user. We believe that structuring of VMs as periodic VMs is a fundamentally sound approach which leads to easier analysis of the worst-cast time behavior of the middleware without incurring any significant performance drawback. A friendly programming interface wrapping the execution support services of TMOSM has also been developed and named the TMO Support Library (TMOSL) [Kim et al. 1999]. It consists of a number of C++ classes and approximates a programming language directly supporting TMO as a basic building block. The programming scheme and supporting tools have been used in a broad range of basic research and application prototyping projects in a number of research organizations and also used in an undergraduate course on RT DC programming at UCI for some years (http://dream.eng.uci.edu/eecs123/SERIOUS.HTM). TMO facilitates a highly abstract programming style without compromising the degree of control over timing precisions of important actions.

2.3 Unmanned Aerial Vehicle (UAV) Control System

An unmanned aerial vehicle (UAV) is an aircraft with no onboard pilot. UAVs can be remote controlled aircrafts (e.g. flown by a pilot at a ground control station), or

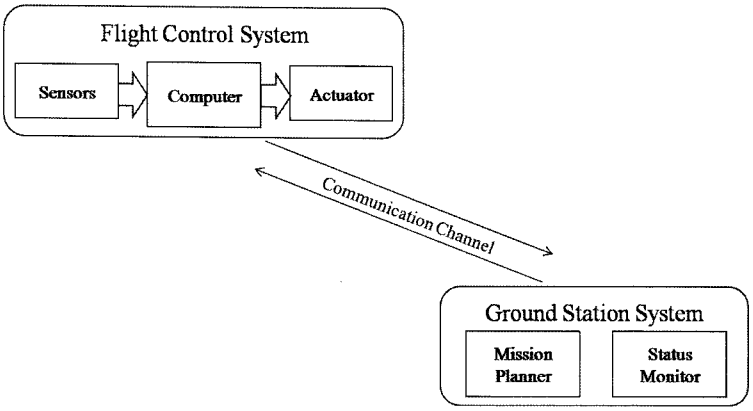


Figure 3. Constitution of the UAV control system.

aircrafts that can fly autonomously with pre-programmed flight plans or dynamically adaptive control systems. Although the UAV control system consists of many components, there are two main components as depicted in Figure 3:

1) An on-board flight control system component of the vehicle, and
2) a ground station component which provides telemetry feedback to the operator and allows him/her to control the aircraft.

In the UAV, the flight control system component (embedded controller) generally consists of three elements - sensors, actuators, and a computer. The control computer interacts with the continuous dynamics of the plant via the sensors and actuators. In our experimental flight control system, an AHRS (Attitude and Heading Reference Systems) and a GPS were used to enable precise control.

2.3.1 *Flight control system component*

(1) Sensors
An AHRS and a GPS receiver were used as the sensors of flight control system. The AHRS was used to provide attitude data. The AHRS was set to generate roll, pitch, and yaw data and sends the data to the control computer, which was connected to the AHRS. The GPS receiver was used to produce geo-metric location data periodically. The GPS provided NMEA standard location data for the control computer in the Flight system component.

(2) Actuators
A servo actuator that was an actuator of servo motors controlled by specific signals in the flight control system. The servo actuator was designed to receive digital servo-control data from the control computer, convert digital data to control signals, and use the control signals to control directly connected servo actuator.

(3) Control computer
The control computer was built using an flight-qualified embedded computer module. In general, this embedded computer has low computing power and

limited resources. Because of this limitation, flight control software has to be optimized to satisfy functional requirement of the flight control system.

Our UAV control system discussed in this paper is designed to fly autonomously with the autopilot. This autopilot requires precise timing of sensor operations. If the timing is badly missed, the UAV would be endangered.

2.3.2 *Ground station component*

(1) Status monitor
In the ground, the operator can monitor the current states of the UAV and issue commands to the running UAV via the ground station component.

(2) Mission planner
The mission planner helps operator to make a plan for the UAV. In general, the plan includes waypoints which are sets of coordinates that identify a point in physical space for the purposes of navigation.

(3) Communication channel
The ground station component and the UAV flight control system component must be connected via communication channels so that they may monitor and command each other.

2.4 FlightGear flight simulator project

FlightGear is an open-source flight simulator project which provides flight simulators running on Windows, Linux and Mac platforms. The goal of the FlightGear project is to create a sophisticated flight simulator framework for use in research or academic environments, for the development and pursuit of other interesting flight simulation ideas, as well as for an end-user application. FlightGear provides a multitude of features such as High Degree of Freedom, Cross Platform, Multiple Flight Dynamic Models, Moderate Hardware Requirements, Extensibility, and Network Access.

The FlightGear aircraft use one of three main data models JSBSim, YAsim, or UIUC. In this paper, we use a JSBSim data model to generates aerodynamic equations of motion in the FlightGear. The JSBSim is a full, six degree-of-freedom flight dynamics model. It provides the capability to specify initial conditions, and to trim the aircraft in several ways prior to startup. Specific integrators can be chosen for propagating each of the translational and rotational velocities and positions.

3. EXPERIMENTAL DESIGN AND IMPLEMENTATION

A high-confidence system like this UAV control system should be designed so that its logical and physical behavior can be easily analyzed and predicted. However, it is also important that timely synchronized behavior between several nodes in distributed enviroment. Therefore, there is a demand for an approach to facilitate both time-triggered design and message-triggered design. To meet this design challenge, we designed our UAV software on the principles of the TMO model.

3.1 Design of an UAV Control Software Based on TMO Model

As depicted in Figure 2, the embedded computer in the flight control system is

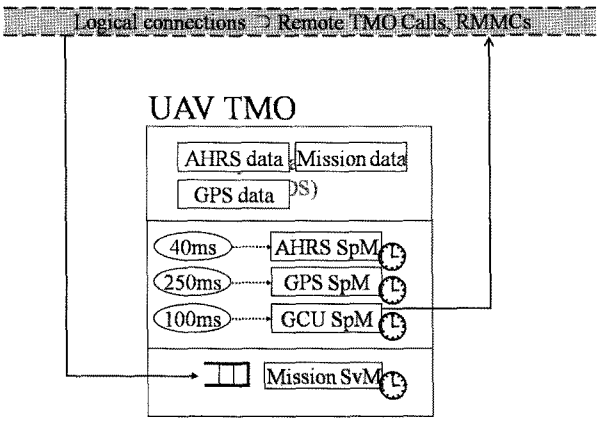


Figure 4. Design of the UAV TMO.

physically connected to sensor and actuator. In addition, it is logically connected to the ground station via the communication channel to communicate with the ground station. In this section, we present the TMO-structured design of the UAV control system.

3.1.1 *UAV TMO: A TMO-structured design of the UAV control software*

The UAV TMO is a design of the embedded computer software in the flight control system. As depicted in Figure 4, the UAV TMO consists of three parts - Time-Triggered Methods, Event-Triggered Methods, and Object Data Store.

(1) Time-Triggered Methods (SpMs): The UAV TMO has three SpMs, named AHRS SpM, GPS SpM and GCU SpM. AHRS SpM and GPS SpM work different period because AHRS and GPS sensor produce sensor data at different rates. In this paper, we assume that AHRS and GPS sensor work every 40 ms and 250 ms respectively. GCU SpM has 100 ms iteration period to report current state of the UAV to the Ground Station TMO using a Gate.

(2) Message-Triggered Methods (SvMs): The UAV TMO has single SvM, named Mission SvM. Mission SvM is invoked when UAV TMO receives command data from Ground Station TMO through the Gate.

(3) Object Data Store (ODS): In the ODS, AHRS, GPS and Command data struc tures are defined.

The detailed functional requirements for each SpM and SvM in the UAV TMO are described in Table I.

3.1.2 *Ground Station TMO: A TMO-structured design of the ground station software*

The Ground Station TMO also consists of three parts as depicted in Figure 5.

(1) Time-Triggered Methods (SpMs): The Ground Station TMO has a single SpM, named Display SpM. Display SpM periodically displays state of the UAV received from the UAV TMO. Display SpM has 100 ms iteration period.

(2) Message-Triggered Methods (SvMs): The Ground Station TMO has two SvMs, one named Mission SvM and the other named Display SvM. Mission SvM is

Table I. Functional requirements of the UAV TMO.

Method	Functions	Deadline
AHRS SpM	1. **Acquire raw attitude data from AHRS sensor every 40 ms.** 2. Analyze and parse raw data and write parsed attitude data to ODS segment. 3. Read attitude data, position data, and command data from each ODS segment. 4. If position data and command data are empty (initial condition), just use AHRS data. 5. If the current position data is not found, produce such data via extrapolation (because GPS sensor internal clock is slower than AHRS sensor). 6. Find flight-path from command data and compare it with the current position data. 7. Generate actuator control signals using PID algorithm to follow given flight-path.	20 ms
GPS SpM	1. **Acquire raw NMEA GPS packet every 250 ms.** 2. Analyze and parse the NMEA GPS packet and write parsed position data to ODS segment.	20 ms
GCU SpM	1. **Read attitude and position data from ODS every 100 ms.** 2. Send attitude and position data to the Ground Station TMO using a Gate.	10 ms
Mission SvM	1. **Receive command data from Ground Station TMO.** 2. Write command data to ODS segment.	10 ms

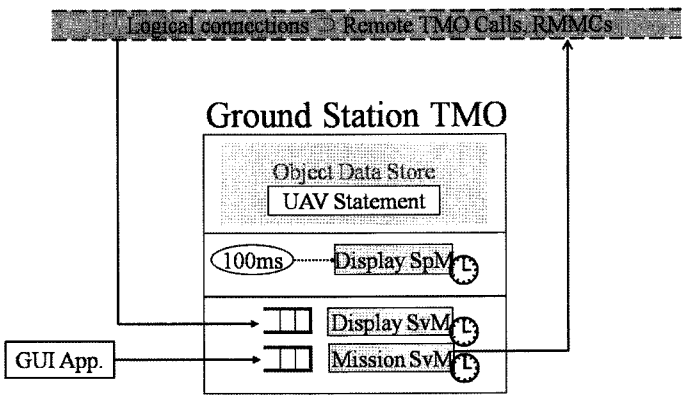


Figure 5. Design of the Ground Station TMO.

invoked by user-application that is GUI-based mission planning software. Display SvM is invoked when Ground Station TMO receives state of the UAV from UAV TMO through the Gate.

(3) Object Data Store (ODS): In the ODS, states of the UAV data structure is defined.

The detailed functional requirements for each SpM and SvM in the Ground

Table II. Functional requirements of the Ground Station TMO

Method	Functions	Deadline
AHRS SpM	1. **Read statement of the UAV from the ODS segment very 40 ms.** 2. Send statement data to the user application for display.	20 ms
Mission SvM	1. Read command data which include waypoints of the UAV from the call parameters supplied by the user application 2. Send the data to the UAV TMO in flight System	20 ms
Display SvM	1. **Read attitude and position data from ODS every 10 ms.** 2. Send attitude and position data to the Ground Station TMO using a Gate.	10 ms

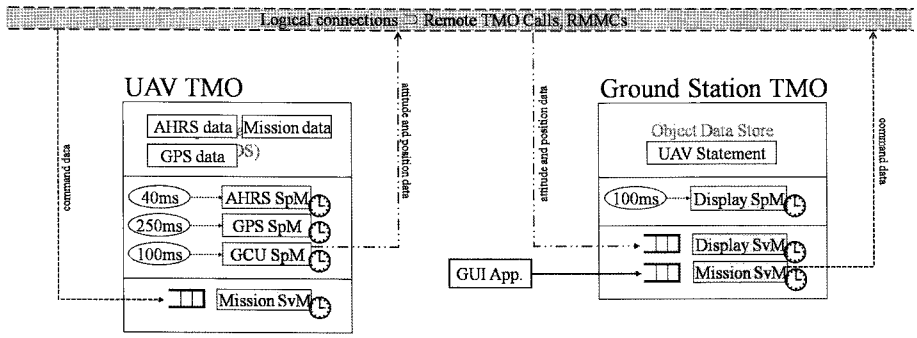


Figure 6. Interaction of the UAV TMO with the Ground Station TMO.

Station TMO are defined in Table II.

3.1.3 *Interaction between UAV TMO and Ground Station TMO*

Figure 6 depicts interactions including data flows between the UAV TMO and the Ground Station TMO. GCU SpM in the UAV TMO requests a service to Display SvM in the Ground Station TMO to display current state of the UAV, while Mission SvM in the Ground Station TMO requests a service to Mission SvM in the UAV TMO to get a new mission data.

3.1.4 *Design extension for multiple-UAV control system*

The previous section described a design methodology for the UAV control system as a standalone system. In this section, we will discuss about design extension to support multiple UAVs in the same manner using TMO model. As mentioned in Table I and Table II, UAVs tasks have very limited period of time (10-20 ms) to process a lot of jobs that include data processing, communication, synchronization, etc. TMO model enables the design of such complex and abstruse system in easily understandable forms. In addition, TMO execution engine will guarantee performance of implemented system without exhaustive testing.

The Global-Time and the RMMC (Real-time Multicast and Memory-Replication Channel) are the most powerful function of TMO model and supported by its

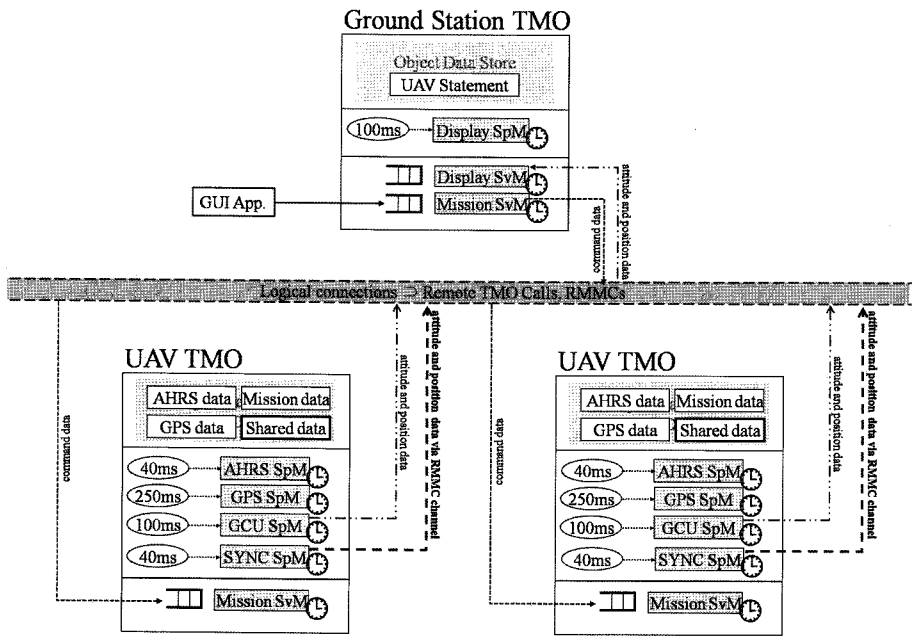


Figure 7. Experimental design of multiple UAV TMOs.

execution engine. Utilizing the Global-Time feature of the TMO, we can simply devise a novel synchronization mechanism that provides inter-location synchronization among multiple receivers distributed through the network. The key of the mechanism is to make the period of processing each packet frame be started as exactly same global time as possible at each receiver side [Jo et al. 2004].

The RMMC scheme is an alternative to the remote method invocation for facilitating interactions among TMOs. Use of RMMCs tends to lead to better efficiency than the use of traditional remote method invocations does in many applications, especially in the area of distributed multicast applications which involve frequent delivery of the same data to more than two participants distributed among multiple nodes [Kim et al. 2005].

In the multiple UAV control system, it is highly required that sharing state of other UAVs periodically to complete cooperational mission properly. To accomplish this additional requirement, we extend our previous design of the UAV TMO experimentally.

As depicted in Figure 7, additional SpM and ODS are added in the design of the UAV TMO. SYNC SpM periodically announces state of current UAV to other nodes which are connected to the same RMMC channel and receives the state of other UAVs from registered RMMC channel. In the ODS, to store other UAV's current state, shared data structure is defined.

Ground Station can choose one or several connection to the UAV TMO for variable mission using the Gate. To increase connection to the UAV TMO, more Gates will require for multiple connection. As discussed in this section, extension of previous system using TMO scheme requires very little modification of overall architecture.

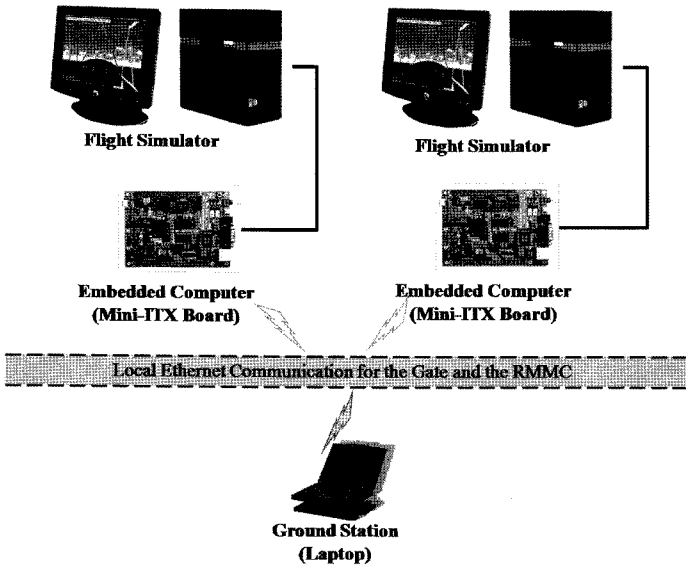


Figure 8. UAV control system architecture for multi-vehicle testing.

We obviously believe that RMMC, Gate and Global Time of TMO are most useful building block to design real-time distributed application like UAV control system.

3.2 Implementation

For our experimental implementation of the UAV control system, mini-ITX embedded computer system and a HILS (Hardware-In-the-Loop Simulation) system were used to compose the UAV flight system virtually. A PID algorithm [Äström 1996; Ledin 2004] was used in our autonomous UAV control system. We used a general-purpose laptop computer as the Ground Station for monitoring the status of the UAV flight control system. Figure 8 presents the hardware structure of the implemented UAV control system for multi-vehicle testing. It was found from this experience of design and implementation of the UAV control system that the TMO-structured real-time computing software in both Flight System and Ground Station turned out to be remarkably easier to read, analyze, and maintain in comparison to the initial version of the software that had been designed 4 years ago and composed of threads, sockets, and thread-priorities.

3.3 Hardware-In-the-Loop Simulation Architecture

In our experiments, we deployed the HILS software centered around FlightGear based on the TMO-HILS architecture [Xu et al. 2007] as depicted in Figure 8. The HILS software is written in C++ and encapsulates the interface to FlightGear. Communications with FlightGear occurs over standard sockets using the user datagram protocol (UDP). The HILS software creates two sockets, one to transmit control data (aileron, elevator, rudder, and throttle) to FlightGear, using the FGNetCtrls class, and the other to receive aircraft state data (airspeed, altitude, position, x/y/z accelerations,

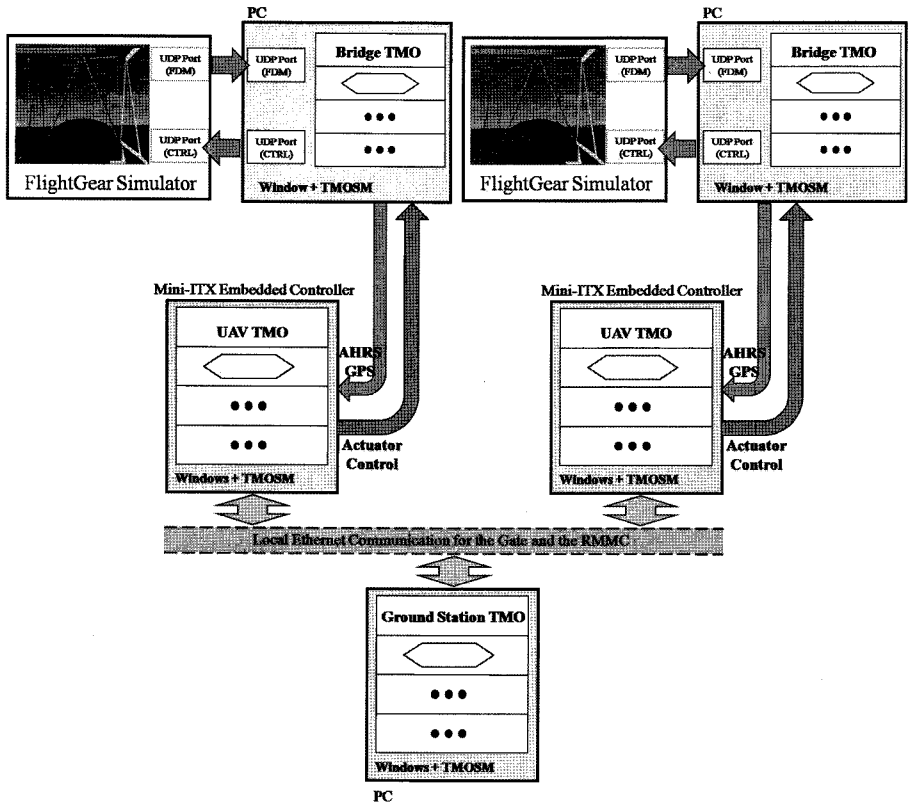


Figure 9. Hardware-In-the-Loop Simulation (HILS) system architecture.

and pitch/roll/yaw rates) from FlightGear, using the FGNetFDM class. Therefore, the FlightGear generates the packet which will be converted by Bridge TMO to the packet of sensors (AHRS, GPS), using FGNetFDM class. In addition, the Bridge TMO converts a control signal of the UAV TMO to a control packet (aileron, elevator, rudder, and throttle) of FlightGear, using FGNetCtrls class.

4. VIRTUAL ENVIRONMENT SIMULATION

4.1 Test with an Environment Simulation

Two important measures of the quality of our experimental helicopter based UAV control system are evaluated in this section:
(1) the degree of synchrony among the UAV nodes and
(2) the performance of stability control.

4.1.1 Degree of synchrony among the UAV nodes

The synchronization of the multiple UAV control system is considered equivalent to the clock synchronization of the UAV nodes.

Therefore, measurements of the clock synchronization must be performed in order

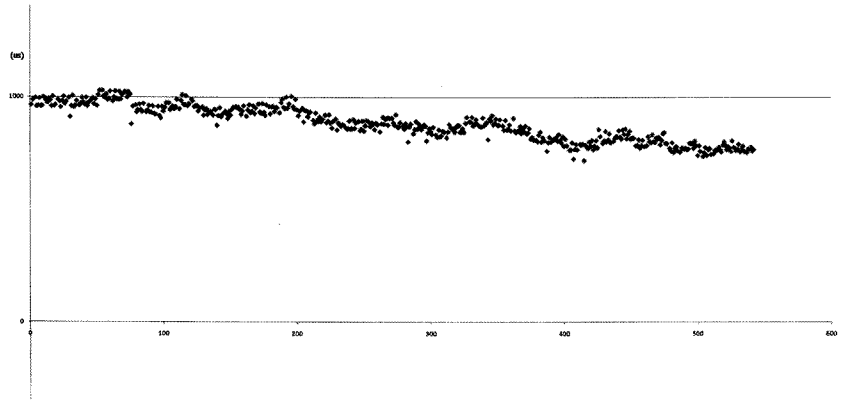


Figure 10. Clock synchronization.

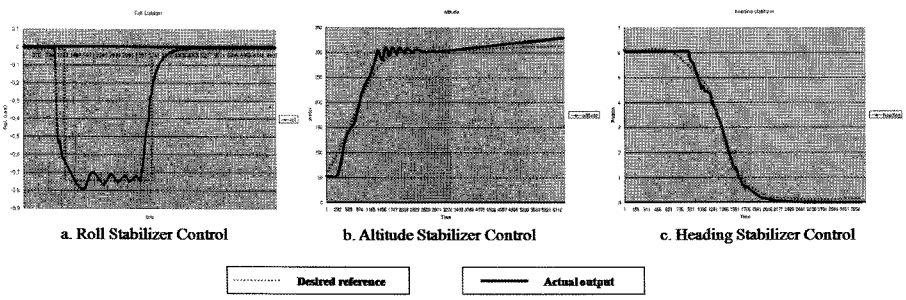


Figure 11. Stability control.

to evaluate the degree of synchrony among the UAV nodes. In this experiment, the clock difference between one UAV node and the other UAV nodes is measured periodically, while the HILS is running. The result is shown in Figure 10.

The clock difference between two UAV nodes range between about 1000 microseconds and about 700 microseconds, and averages about 850 microseconds. Considering the period of most important SpM (AHRS SpM), this result is enough to work sychronously.

4.1.2 Performance of stability control

In this experiment, we tested roll, altitude, and heading stabilizer control of the single UAV control system on the HILS. Figure 11 shows the results. The desired control references, which are generated by JSBSIM in FlightGear as an ideal output of control, are displayed as dotted lines and the actual responses of the embedded controller in the UAV are displayed as bold lines. These response data were obtained while the UAV in simulation changed the altitude and heading into the direction toward the destination. Figure 9 has three different graphs which show changes in the stabilizer of roll, altitude, and heading.

When the UAV has to change heading direction toward the destination (See Figure 9-c), roll and pitch(altitude stabilizer) values will also be changed until heading is set in the direction toward the destination (See Figure 9-a, b). During this

period, the UAV encounters an unstable situation, and the embedded controller should adjust the attitude of the UAV for stable flight. The results of each stabilizer control indicate that the actual response data obtained are close to the desired references for stable flight. By comparing the figures, we can conclude that the exhibited behavior of the simulated Flight System is close to the desired references. These results attest to the accuracy of our embedded control system implemented with the TMO structuring techniques and tools and the usefulness of the hardware-inthe-loop simulator.

5. CONCLUSION AND FUTURE WORK

In this paper, we presented our experimental application of the TMO structuring scheme to the design and implementation of multiple UAV control system. The Flight System and the Ground Station were designed by application of the TMO scheme and implemented with TMOSM. We tested our UAV control system by means of flight simulation test. We could confirm a reasonable-level performance of roll, altitude, and heading stabilizer control from flight simulations. We also presented the HIL simulator for the UAV that was used to obtain test flight results. Therefore, these experimental research results clearly indicate the promising nature of the TMO structuring scheme in design and implementation of challenging real-time distributed computing software such as those needed in the UAV control systems. The advantages of the TMO scheme over conventional low-level programming approaches involving composition of software with threads, sockets, and thread-priorities seem quite significant. Among several areas in which the UAV control system could be improved, fault-tolerance in UAV control systems is an item of top-priority to us for tackling in the near future research.

ACKNOWLEDGMENTS

This research was supported by the MKE (Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA (Institute for Information Technology Advancement). (IITA-2008-C1090-0804-0015).

REFERENCES

ÄSTRÖM, K. J. 1996. *PID Control in the Control Handbook.* William S. Levine (Boca Raton, FL: CRC Press).

DOHERTY, P., P. HASLUM, F. HEINTZ, T. MERZ, P. NYBLOM, T. PERSSON, AND B. WINGMAN. 2004. A distributed architecture for intelligent unmanned aerial vehicle experimentation. In *Distributed Autonomous Robotic Systems 6.* Proceedings of the 7th International Symposium on Distributed Autonomous Robotic Systems., Springer-Verlag.

HOROWITZ, B., J. LIEBMAN, C. MA, T. J. KOO, A. SANGIOVANNI-VINCENTELLI, AND S. S. SASTRY. 2003. Platform-based embedded software design and system integration for autonomous vehicles. *IEEE Transactions 91*, 111.

IPPOLITO, C. 2005. An autonomous autopilot control system design for small-scale uavs. Internal Report of CMIL EAV-20051016, University of Carnegie Mellon.

JO, E.-H., D.-H. KIM, H. PARK, AND M.-H. KIM. 2004. Globally synchronized multimedia streaming architecture based on time-triggered and message-triggered object model. In

LNCS 3320. Parallel and Distributed Computing: Applications and Technologies, Springer, Berlin, Singapore, 891.

KIM, H.-J., S.-H. PARK, J.-G. KIM, M.-H. KIM, AND K.-W. RIM. 2002. Tmo-linux: A linux-based real-time operating system supporting execution of tmos. In *ISORC '02: Proceedings of the Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing.* IEEE Computer Society, Washington, DC, USA, 288.

KIM, K. H. K. 1997. Object structures for real-time systems and simulators. *Computer 30*(8): 62–70.

KIM, K. H. K. 2000. Apis for real-time distributed object programming. *Computer 33*(6):72–80.

KIM, K. H. K., M. ISHIDA, AND J. LIU. 1999. An e'cient middleware architecture supporting time-triggered message-triggered objects and an nt-based implementation. In *ISORC '99: Proceedings of the 2nd IEEE International Symposium on Object-Oriented Real-Time Distributed Computing.* IEEE Computer Society, Washington, DC, USA, 54.

KIM, K. H. K., Y. LI, S. LIU, M. H. KIM, AND D.-H. KIM. 2005. Rmmc programming model and support execution engine in the tmo programming scheme. In *ISORC '05: Proceedings of the Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing.* IEEE Computer Society, Washington, DC, USA, 34–43.

LEDIN, J. 2004. *Embedded Control Systems in C/C++: An Introduction for Software Developers Using MATLAB.* CMP Books.

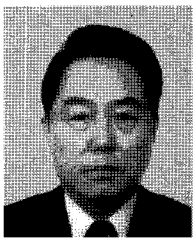
PARK, H., M.-H. KIM, C.-H. CHANG, K. KIM, J.-G. KIM, AND D.-H. KIM. 2007. Design and experimental validation of uav control system software based on the tmo structuring scheme. In *LNCS 4761.* International Federation for Information Processing, Springer Berlin/Heidelberg, Santorini Island, Greece, 192–201.

SORTON, E. F. AND S. HAMMAKER. 2005. Simulated ight testing of an autonomous unmanned aerial vehicle using ightgear. In *AIAA 2005-7083.* Institute for Scienti'c Research, Inc, American Institute of Aeronautics and Astronautics, Inc., Arlington, Virginia.

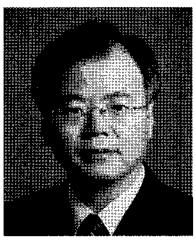
XU, J., H. PARK, K. JEONG, AND C.-H. CHANG. 2007. Tmo-hils architeture for real-time control system. In *ICCAS 2007.* Institude of Control, Robotics and Systems, ICROS, Seoul, Korea, 1855–1861.

Hansol Park received BS in Aerospace Engineering and MS in Computer Science from Konkuk University respectively in 2004 and 2006. During his graduate study, he conducted research on the UAV control system, and real-time simulation. He is currently a researcher in the Agency for Defense Development. Recently he has investigated avionics system, and autonomous control.

Doo-Hyun Kim is Associate Professor of Internet and Media Engineering at the Konkuk University. He received MS and Ph.D. degree from the Computer Science at the Korea Advanced Institute of Science and Technology respectively in 1987 and 2003. He also worked for ETRI as principal research engineer in 1997-2003. His research interests include TMO-based Embedded Software, Sensor OS, Real-Time Computing in UAV, Linux and Open Source S/W Wngineering.



Jung-Guk Kim is Professor of Computer Engineering at the Hankuk University of Foreign Studies. He received a Ph.D. degree from the Computer Science at the Korea Advanced Institute of Science and Technology in 1986. His research interests include real-time operating system, and embedded system.



Chun-Hyon Chang is Professor of Computer Science and Engineering at the Konkuk University. He received MS and Ph.D. degree from the Computer Science at the Korea Advanced Institute of Science and Technology respectively in 1979 and 1985. His research interests include Programming Environment, TMO-based Embedded Software, Real-Time Computing in UAV.