Technical Paper
J. Astron. Space Sci. 27(4), 407-412 (2010)
DOI: 10.5140/JASS.2010.27.4.407

JASS Journal of Astronomy and Space Sciences

SMI Compatible Simulation Scheduler Design for Reuse of Model Complying with SMP Standard

Cheol-Hea Koo[1†], Hoon-Hee Lee[2], and Yee-Jin Cheon[1]

[1]Satellite Flight Software Department, Satellite Technology Division, Korea Aerospace Research Institute, Daejeon 305-333 Korea
[2]Geostationary Satellite Operation Department, Satellite Technology Division, Korea Aerospace Research Institute, Daejeon 305-333 Korea

Software reusability is one of key factors which impacts cost and schedule on a software development project. It is very crucial also in satellite simulator development since there are many commercial simulator models related to satellite and dynamics. If these models can be used in another simulator platform, great deal of confidence and cost/schedule reduction would be achieved. Simulation model portability (SMP) is maintained by European Space Agency and many models compatible with SMP/simulation model interface (SMI) are available. Korea Aerospace Research Institute (KARI) is developing hardware abstraction layer (HAL) supported satellite simulator to verify on-board software of satellite. From above reasons, KARI wants to port these SMI compatible models to the HAL supported satellite simulator. To port these SMI compatible models to the HAL supported satellite simulator, simulation scheduler is preliminary designed according to the SMI standard.

Keywords: simulation model portability, simulation model interface, simulated software test bench, simSTB, scheduler, simulator

1. INTRODUCTION

Satellite simulation environment has very important role during software and system verification and testing phase since it is cost effective way to do it. The usage of simulator on software testing and system verification is being increased continuously. For example, in communication, ocean, and meteorological satellite (COMS) program, launched at 27 June 2010, various simulators were used at software and system verification at each phase of software testing, satellite dynamics verification, satellite engineering test bed test, and even satellite integration test phase. In these simulators, various simulation models are integrated inside for their specific simulation capability. For example, from telemetry and telecommand model as on-board computer component to sensor and dynamics model as satellite analysis model, almost everything on satellite can be modeled.

It should be noted that purchasing these models which are already proven may be the most effective way to development simulator rather than starting with scratch from everything, e.g. simulator platform, simulator/man interface, simulator/model interface, and finally model itself. To use pre-developed simulator model, simulator/model interface is very important to maintain the reusability through various simulator platforms.

EuroSim and SIMSAT is widely known as satellite simulator platform and both are being used by European spacecraft development companies and European Space Agency (ESA)/European Space Operations Centre

Received Sep 10, 2010 **Revised** Oct 14, 2010 **Accepted** Nov 22, 2010
†**Corresponding Author**
E-mail: chkoo@kari.re.kr
Tel: +82-42-860-2145 Fax: +82-42-860-2007

(ESOC)/European Space Research and Technology Centre (ESTEC). They are very powerful platform and have many third parties who provide various simulator models. VEGA and Terma are examples of these third parties.

Unfortunately, there are cases that another specific functionality which is not supported by EuroSim or SIMSAT, is requested. For example if we need to integrate several simulators through ethernet or something else, native function of these simulator platforms is not sufficient to do it since we don't know the detail mechanism of these simulator platforms.

So, simplified and flexible simulator platform to meet the specific requirements has been requested frequently since there are numerous phases and levels on software and system test/verification of satellite. The numerous phases and levels bring various functional requirements about satellite simulator environment.

2. SIMULATION MODEL INTERFACE/SIMULATION MODEL PORTABILITY

Even though new simulator platform is required, existing simulator models shall be used as much as can with minimal design or source code change. So interface between simulator platform and simulator model is requested to maintain consistency of functionality of simulator model when simulator platform is changed.

Simulation model portability (SMP) is one of the good candidates of the interface between simulator platform and simulator model (ESA 2002). SMP is introduced by ESA and focuses on maintaining the interface consistency with various simulator platform, various development languages, and various operating systems. Those environments which shall be compatible under SMP would be summarized as follows:

- Simulator platform: EuroSim, SIMSAT, Rose and etc.
- Development languages: Java, Ada, C/C++, Fortran
- Operating systems: MS Windows, Linux

SMP standard does not describe detail implementation of the interface, but describes only abstraction interface application programming interface (API) of simulation model interface (SMI).

As depicted in Fig. 1, simulator models are interfaced with simulation platform through SMI APIs. The dotted line in Fig. 1 represents inside relationship between models and solid line in Fig. 1 represents interface with simulator platform via SMI. Through SMI APIs, model can register its service (function) and data to simulator model manager as depicted in Fig. 2. Once the services and data are registered to a simulator platform, the services and data can be used by the simulator core function, e.g. scheduler or javascript as test script from user. The core function is to manage the simulator periodic/aperiodic event thanks to scheduler which is composed of various functions supporting real-time requests.

3. PRELIMINARY DESIGN OF SIMULATION SCHEDULER

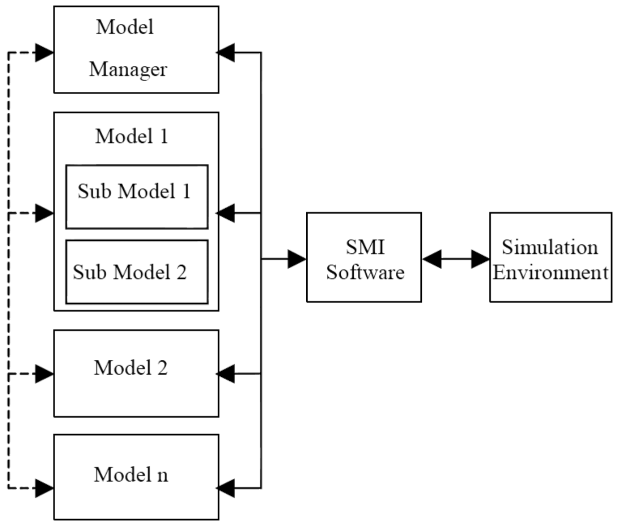


Fig. 1. Simulation model portability architecture (ESA 2002).

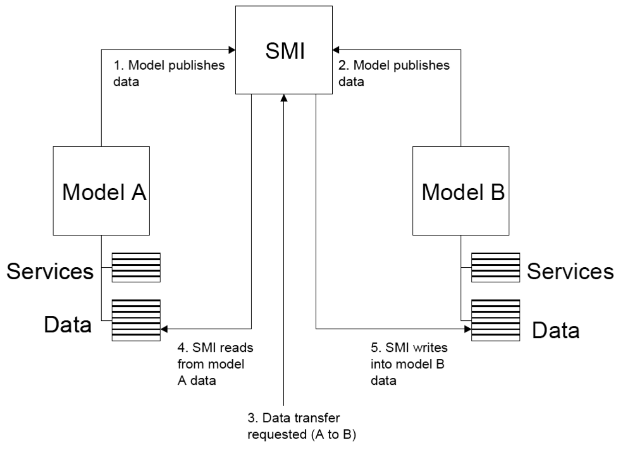

SMI
1. Model publishes data
2. Model publishes data
Model A
Model B
Services
Data
Services
Data
4. SMI reads from model A data
5. SMI writes into model B data
3. Data transfer requested (A to B)

Fig. 2. Model data transfer (ESA 2002).

Simulation scheduler is the most frequently called function in simulation platform. Scheduler distributes time resources such as simulation time and processor emulation time, and manages simulation event. The basic operation of scheduler is that it remembers simulation event, continues the process of processor emulator until the shortest event time is elapsed, and then pauses the processor emulator and jumps the simulation event handler to service the functionality of event. When the processing of an event is completed, it returns to a paused processing of a processor emulator.

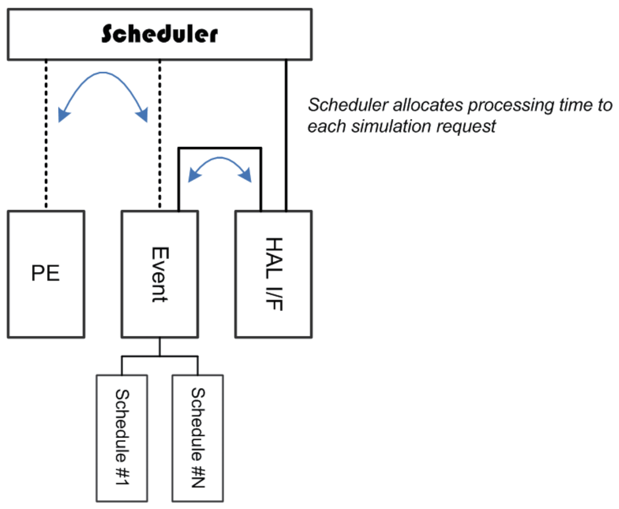

Scheduler
Scheduler allocates processing time to each simulation request
PE
Event
HAL I/F
Schedule #1
Schedule #N

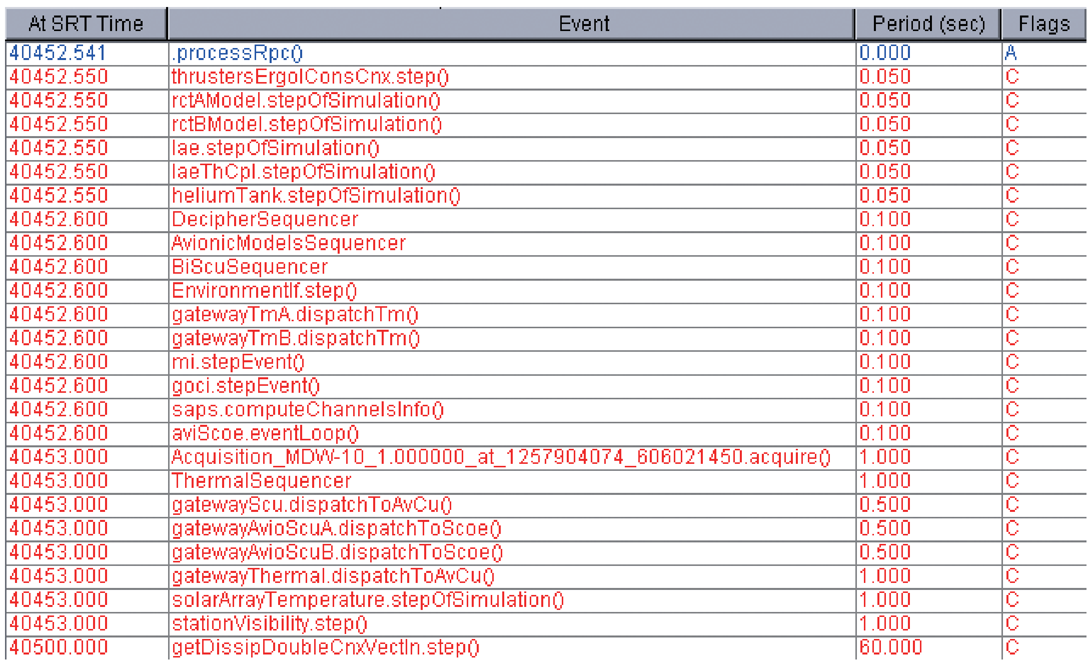
Fig. 3. Scheduler interface to processor emulator and event.

To maintain the compatibility to SMI/SMP of existing simulator models, simulation scheduler is designed to provide simplified SMI APIs to further simulator model. This simulation scheduler shall have minimal dependency with development platform, e.g. operating system and development tools.

As shown in Fig. 3, scheduler allocates processing time to each simulation request, e.g., processor emulator and simulation event. The solid dot line means that this allocation is based on "Soft Real-Time" scheduling concept and the solid line connecting between scheduler and hardware abstraction layer (HAL) I/F means that this allocation is based on "Non Real-Time" scheduling concept. Fig. 4 represents a schedule as a part of scheduler, which is the list of simulation events that happen in SIMWARE of COMS dynamic satellite simulator software. Those events are expected which models will run and when models will run.

Under "Soft Real-Time" scheduling concept, scheduler tries to make the simulation time same with real world time, i.e., Zulu Time, as close as it can. Scheduler adjusts the simulation time in order to prevent it from deviating from Zulu Time too much.

Under "Non Real-Time" scheduling concept, scheduler does not manage the simulation time since it is almost



At SRT Time	Event	Period (sec)	Flags
40452.541	.processRpc()	0.000	A
40452.550	thrustersErgolConsCnx.step()	0.050	C
40452.550	rctAModel.stepOfSimulation()	0.050	C
40452.550	rctBModel.stepOfSimulation()	0.050	C
40452.550	lae.stepOfSimulation()	0.050	C
40452.550	laeThCpl.stepOfSimulation()	0.050	C
40452.550	heliumTank.stepOfSimulation()	0.050	C
40452.600	DecipherSequencer	0.100	C
40452.600	AvionicModelsSequencer	0.100	C
40452.600	BiScuSequencer	0.100	C
40452.600	EnvironmentIf.step()	0.100	C
40452.600	gatewayTmA.dispatchTm()	0.100	C
40452.600	gatewayTmB.dispatchTm()	0.100	C
40452.600	mi.stepEvent()	0.100	C
40452.600	goci.stepEvent()	0.100	C
40452.600	saps.computeChannelsInfo()	0.100	C
40452.600	aviScoe.eventLoop()	0.100	C
40453.000	Acquisition_MDW-10_1.000000_at_1257904074_606021450.acquire()	1.000	C
40453.000	ThermalSequencer	1.000	C
40453.000	gatewayScu.dispatchToAvCu()	0.500	C
40453.000	gatewayAvioScuA.dispatchToScoe()	0.500	C
40453.000	gatewayAvioScuB.dispatchToScoe()	0.500	C
40453.000	gatewayThermal.dispatchToAvCu()	1.000	C
40453.000	solarArrayTemperature.stepOfSimulation()	1.000	C
40453.000	stationVisibility.step()	1.000	C
40500.000	getDissipDoubleCnxVectIn.step()	60.000	C

Fig. 4. Example of schedule on COMS dynamic satellite simulator software simulator.

impossible to handle correctly the delay introduced by HAL I/F. So, scheduler will pause whole simulation time and wait until a feedback is delivered from HAL I/F.

Table 1. Summary of SMI API related to scheduler.

SMI API name	Parameters		Comment
SMI Publish Service()	Const ObjectID_T Const ServiceName_t Const Service_t ServiceID_t	ObjectID ServiceName Service *ServiceID	 Event entry point
SMI ExtSchedule Event()	Const Unsigned64_t Const Unsigned64_t Const ObjectID_t Const ServiceID_t Const Parameter_t EventID_t	CycleTime DeltaTime ObjectID ServiceID *pParameters *pEventID	Period Initial delay
SMI ExtDeleteEvent()	Const EventID_t	EventID	

SMI: simulation model interface, API: application programming interface.

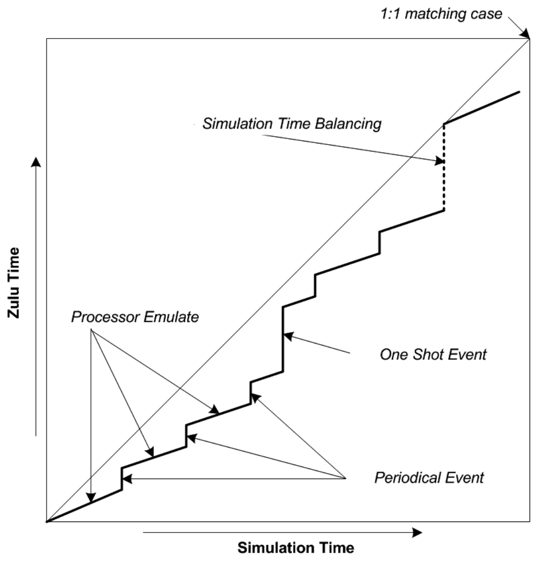


Fig. 5. Scheduler load balancing mechanism.

Simulation model consists of service and data and these service and data can be published to simulator platform thanks to SMI APIs like SMIPublishService() API and SMIPublishData() API. Especially service can be used by scheduler for the target entry point of event handler. Scheduled event can be registered by SMIExtScheduleEvent() API with Cycle Time and Delta Time parameters. So, the event handler published by SMIPublishService() API can be called periodically or aperiodically if Cycle Time is assigned or not.

The prototypes of these APIs which are implemented in current porting are summarized as shown in Table 1.

To hold these schedule information, simple data structure based on the above parameter lists shown in Table 1 is introduced. And, special data for Remain Time is added to the data structure. Remain Time designates a time which is left to an event from current simulation time. Remain Time is crucial concept since scheduler works along with processor emulator but with simulator itself. So, minimal thread collaboration is introduced for this scheduler design. Only two threads are necessary. One is for scheduler, and the other is for processor emulator handling. An event which has the shortest Remain Time is supposed to be called after passing the shortest Remain Time.

Because current processing power of personal computer or workstation is much higher than satellite onboard computer, the simulation time is much smaller than real world time. As the result, a user can see the simulation as like quick moving picture. Of course it may be helpful to cover a long time simulation period of low activity quickly. As shown in Fig. 5, load balancing can be used to boost up or slow down the simulation period which shown to user.

So, scheduler operates as shown in Fig. 6 (ESA 2008). Scheduler services the periodical or aperiodical event if applicable. Otherwise processor emulator dominantly possess the simulation time until new event is available.

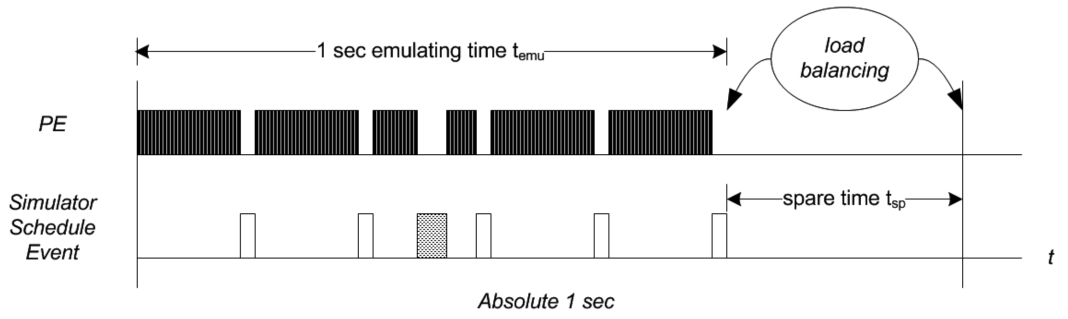


Fig. 6. Scheduler operation.

Actually processing time of each event is not reflected at simulation time. If simulation time is too deviated to real world time, Zulu Time, simulation is paused thanks to simulation time balancing (load balancing).

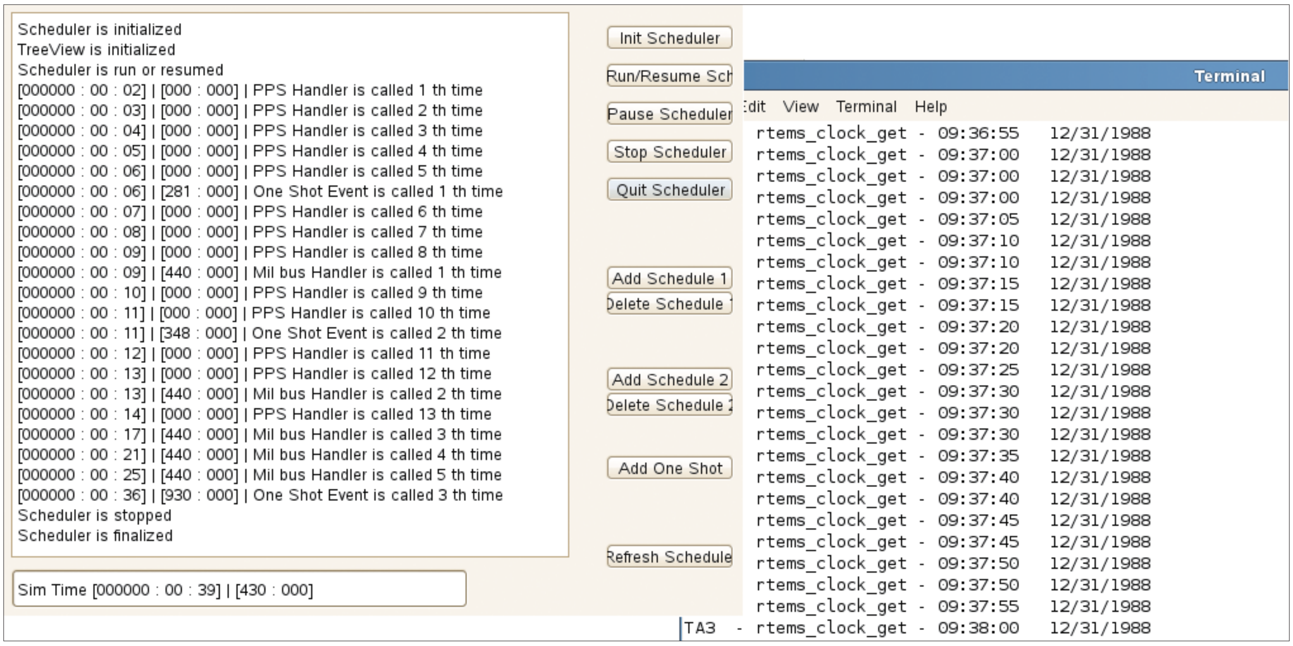
4. SIMULATION SCHEDULER DEMONSTRATION

Prototype of simulation scheduler is developing with Qt and C++ programming language on Linux. Simplified but full functional SMIPublishService(), SMIExtScheduleEvent() and SMIExtDeleteEvent() APIs are implemented. Also simulator model object management simulator module is also implemented since model object is important to call the event handler published by SMIPublishService() API.

A demo simulator which consists of following item or processing is proposed.

- Simulator model registration
- Simulator model initialization
  - • Add several services through SMIPublish Service() API
    - Pusle per second (PPS) handler
    - Mil bus handler
  - • Set schedul through SMIExtScheduleEvent () API
    - PPS handler, set to 1 sec cycle
    - Mil bus handler, set to 4 sec cycle with 4 sec delay
- Scheduler initialization
- Simulation start
  - • Add one shot event
    - Just with 30 ms delay
  - • Delete scheduled event through SMIExt DeleteEvent() API
    - First delete PPS event
    - Second delete Mil bus event after 11 sec from deleting PPS event
- Simulation stop
- Scheduler stop

In this demo simulator, the tested simulator model is the exact same with simulator models which are working on SIMSAT v 4.0. The only difference is TSIM from Aeroflex Gaisler which is used for processor emulator for ERC32 sparc processor. It means there is no dependency to processor emulator for scheduler implementation. Fig. 7 presents a demo of SMI scheduler with example RTEMS program running with TSIM for ERC32. The demonstration is performed by above scheduled items. In left window in Fig. 7, scheduler event is being logged. In right window in Fig. 7, running message from the example RTEMS program is being displayed. So it can be conclud-



```
Scheduler is initialized
TreeView is initialized
Scheduler is run or resumed
[000000 : 00 : 02] | [000 : 000] | PPS Handler is called 1 th time
[000000 : 00 : 03] | [000 : 000] | PPS Handler is called 2 th time
[000000 : 00 : 04] | [000 : 000] | PPS Handler is called 3 th time
[000000 : 00 : 05] | [000 : 000] | PPS Handler is called 4 th time
[000000 : 00 : 06] | [000 : 000] | PPS Handler is called 5 th time
[000000 : 00 : 06] | [281 : 000] | One Shot Event is called 1 th time
[000000 : 00 : 07] | [000 : 000] | PPS Handler is called 6 th time
[000000 : 00 : 08] | [000 : 000] | PPS Handler is called 7 th time
[000000 : 00 : 09] | [000 : 000] | PPS Handler is called 8 th time
[000000 : 00 : 09] | [440 : 000] | Mil bus Handler is called 1 th time
[000000 : 00 : 10] | [000 : 000] | PPS Handler is called 9 th time
[000000 : 00 : 11] | [000 : 000] | PPS Handler is called 10 th time
[000000 : 00 : 11] | [348 : 000] | One Shot Event is called 2 th time
[000000 : 00 : 12] | [000 : 000] | PPS Handler is called 11 th time
[000000 : 00 : 13] | [000 : 000] | PPS Handler is called 12 th time
[000000 : 00 : 13] | [440 : 000] | Mil bus Handler is called 2 th time
[000000 : 00 : 14] | [000 : 000] | PPS Handler is called 13 th time
[000000 : 00 : 17] | [440 : 000] | Mil bus Handler is called 3 th time
[000000 : 00 : 21] | [440 : 000] | Mil bus Handler is called 4 th time
[000000 : 00 : 25] | [440 : 000] | Mil bus Handler is called 5 th time
[000000 : 00 : 36] | [930 : 000] | One Shot Event is called 3 th time
Scheduler is stopped
Scheduler is finalized

Sim Time [000000 : 00 : 39] | [430 : 000]
```

```
  rtems_clock_get - 09:36:55   12/31/1988
  rtems_clock_get - 09:37:00   12/31/1988
  rtems_clock_get - 09:37:00   12/31/1988
  rtems_clock_get - 09:37:00   12/31/1988
  rtems_clock_get - 09:37:05   12/31/1988
  rtems_clock_get - 09:37:10   12/31/1988
  rtems_clock_get - 09:37:10   12/31/1988
  rtems_clock_get - 09:37:15   12/31/1988
  rtems_clock_get - 09:37:15   12/31/1988
  rtems_clock_get - 09:37:20   12/31/1988
  rtems_clock_get - 09:37:20   12/31/1988
  rtems_clock_get - 09:37:25   12/31/1988
  rtems_clock_get - 09:37:30   12/31/1988
  rtems_clock_get - 09:37:30   12/31/1988
  rtems_clock_get - 09:37:30   12/31/1988
  rtems_clock_get - 09:37:35   12/31/1988
  rtems_clock_get - 09:37:40   12/31/1988
  rtems_clock_get - 09:37:40   12/31/1988
  rtems_clock_get - 09:37:45   12/31/1988
  rtems_clock_get - 09:37:45   12/31/1988
  rtems_clock_get - 09:37:50   12/31/1988
  rtems_clock_get - 09:37:50   12/31/1988
  rtems_clock_get - 09:37:55   12/31/1988
TA3 - rtems_clock_get - 09:38:00   12/31/1988
```

Fig. 7. Scheduler demonstration.

ed that the demo simulator well represents the scheduler functionality of SIMSAT for the view of compatibility on SMP/SMI under simplified demonstration purpose.

5. CONCLUSIONS

The purpose of developing simulator scheduler is to develop a mission specific simulator platform so as to cover new mission specific requirements more quickly in addition to reuse heritage simulator model compatible with SMP/SMI.

KARI has been developing simulated software test bench (STB) under SIMSAT 4.0 with simulator models which are designed to be compatible with SMP/SMI. Since recently a mission specific requirement is introduced, HAL supported satellite simulator is being considered to be developed (Koo & Lee 2009). It is checked that the proven simulator models can be used if a simulator platform supports SMP/SMI and commercial simulator models can be integrated in the simulator platform.

It would be a meaningful conclusion that the support to the SMP/SMI can reduce risk, cost, development time for the manufacture of the HAL supported satellite simulator.

REFERENCES

European Space Agency (ESA) 2002, EWP-2080

European Space Agency (ESA) 2008, EGOS-SIM-SIM-SUM-1001

Koo, C. H. & Lee, S. K. 2009, Approach to the Use of Simulated Software Test Bench in Integration Test of Flight Software, INTELEC 2009