SDYA: A Real Time and Distributed Software Verification Infrastructure for Validating Flight Software (On-Board Software) at System Integration Laboratory.

Uğur Melih Sürme(1), Engin Öztuna (2) , Orhan Uğurlu(3) , Uğur Çakır(4) , Samet Nargül(5) , Kadriye Güçlü(6)

(1,2,3,4,5)TAI, Turkish Aerospace Industries, Inc.(TURKEY)
Fethiye Mahallesi, Havacılık Bulvarı No:17
06980 Kazan-ANKARA / TURKEY
Email:msurme@tai.com.tr, eoztuna@tai.com.tr, ougurlu@tai.com.tr, ucakir@tai.com.tr, snargul@tai.com.tr,kguclu@tai.com.tr

ABSTRACT

System Integration Laboratory (SIL) Verification Software Infrastructure (“SDYA”) is developed by Turkish Aerospace Industries (TAI) Software Team. It is currently being used in the development of simulation software required for system and software verification (testing and integration) at System Integration Laboratory for safety critical/reliable flight software (On-Board Data Handling Software) and surrounding avionic equipment for various platforms like manned/unmanned aerial vehicles and satellite systems. ECSS-E-TM-40-07 (SMP-2): "Simulation Modelling Platform" standard compliance is taken as a design consideration, and SDYA has been developed in accordance with CMMI Level 3 compliant TAI processes. Since the newly developed flight software to be tested will have to comply with either “ECSS: ECSS-E-ST-40C: Space Engineering Software” or “RTCA DO-178B: Software Considerations in Airborne Systems and Equipment Certification”, SDYA is designed and developed satisfying both standards and related handbooks Verification Tool/Tool Qualification objectives. SDYA also provides an automated test infrastructure making it easier to conduct regressions when changes occur. SIL Verification Software developed using SDYA infrastructure plays an important role in safety critical/reliable software development projects by supporting the detection and resolution of critical errors in the early stages of the projects, hence, reduces technical risks and costs while shortening the development time.

Key Words: ECSS, SMP, Flight Software, Simulation, Real Time, Distributed, Automated Test, SIL, CMMI, DO-178B, Software Verification Tool, Tool Qualification, MIL-STD-1553, SpaceWire

INTRODUCTION

System Integration Laboratory Verification Software (SDYA) is a generic, real time, distributed simulation infrastructure (engine) software developed by TAI for the verification of Flight Software, Ground Station Command and Control Software within a simulated laboratory environment. It replaces the real avionics equipment, and mimics their functional behavior, thus communicates with Flight Software (On-Board Data Handling Software), and Ground Station Command and Control Software through real avionics interfaces (MIL-STD-1553, Serial, SpaceWire, Can Bus, etc.). It is used to apply integrated test scenarios in system integration laboratory. Satellite System Integration Laboratory at TAI Facilities is shown in Fig.1.

Fig. 1 Satellite System Integration Laboratory at TAI Facilities

Because of TAI's platform integration responsibilities as being the main contractor in many satellite projects, TAI is developing verification software for testing its own indigenously developed Flight Software. Verification at SIL helps in detecting errors related with the avionics system & architecture, interfaces, and Flight Software, thus reduces project risks and cost, while improving the quality of avionic software during the integration and testing activities [1].

In order to meet the challenges of these complex and highly interoperable systems, the SIL has become a key component in the development process, and provides a test environment that is a cross between a pure simulation and the final system.

SDYA has been developed in accordance with CMMI Maturity Level-3 compliant TAI Processes. The Verification Tool requirements, which are explained in safety critical/reliable software development standards such as ESA ECSS, and RTCA DO-178B Tool Qualification, are fully satisfied. SDYA also contains a test infrastructure consisting of both manual and automated tests developed for its own qualification, making it easy to conduct even full regression testing when changes occur.

Following its own qualification tests, SDYA has also been successfully qualified for "ANKA" Unmanned Aerial Vehicle (UAV) Program (An Advanced Medium Altitude Long Endurance (MALE) class UAV System) in Q2, 2014 following the specific ANKA models' development and customization. "ANKA" performs day and night, all-weather reconnaissance, target detection / identification and intelligence missions with its EO/IR and SAR payloads, featuring autonomous flight capability including Automatic Take-off and Landing.

SDYA can be tailored and used as a development or verification tool for flight software and avionics system verification needs in Software Verification Facility (SVF) [2], during Functional Chain Validation (FCV), or in Dynamic Satellite Simulator Development. It can also be used as a support tool for verification of satellite on-board control procedures, validation of satellite control centre, and satellite operator training. SMP-2 compliance is taken as a design consideration for utilization in space domain.

SDYA OVERVIEW

SDYA provides a simulation based verification infrastructure for the validation of the Flight Software and can be utilized in the following critical activities;

- Integration of avionics software/hardware
- Verification of software and hardware interfaces
- Simulation of avionic equipment and environment (simulation model development)
- Avionics hardware and software acceptance testing
- Avionic equipment fault detection

In order to satisfy these goals, selection of critical technologies in the SDYA development process has been realized by the application of Decision Analysis and Resolution (DAR) Process [3]. For instance, Operating System decision has been concluded by the selection of Windows and Linux for SDYA implementations. One of the major deriving factors (DAR criteria) was cross platform development and execution needs of SDYA programmers and potential users. Cross platform development evolves from portable coding practices. Portable source code will compile and execute on Linux and Windows even if underlying hardware is different. Thus, it gives the programmers and/or users the ability to run the same software on different operating systems and hardware configurations.

Selected technologies in the development of SDYA, and its component based architecture are shown in Fig.2.

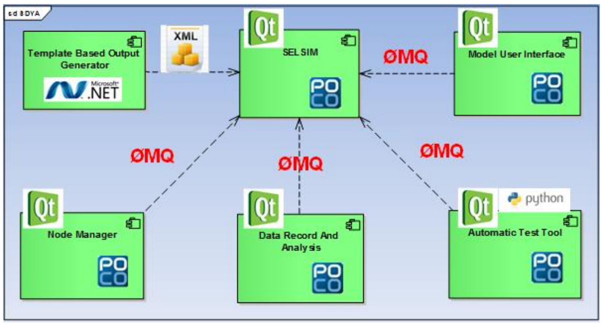
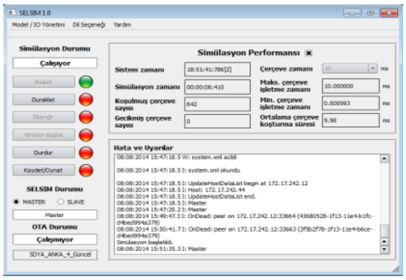


Fig. 2 SDYA: Selected Technologies and Component Based Architecture

SDYA infrastructure consists of the following software modules;

SELSIM: It contains all models required within a simulation and controls the scheduling of models. It publishes/subscribes information of simulation objects to the outside world i.e. Model User Interface. It is able to communicate over the real equipment data buses. SELSIM user interface is shown in Fig.3.



Simülasyon Durumu: Çalışıyor

Simülasyon Performansı

Sistem zamanı	18:51:41:786[Z]	Çerçeve zamanı	10	ms
Simülasyon zamanı	00:00:06:410	Maks. çerçeve işletme zamanı	10.000000	ms
Koşulmuş çerçeve sayısı	642	Min. çerçeve işletme zamanı	0.000993	ms
Gecikmiş çerçeve sayısı	0	Ortalama çerçeve koşturma süresi	9.98	ms

SELSIM Durumu: [✓ MASTER] SLAVE
Master

OTA Durumu: Çalışmıyor

SDYA_ANKA_4_Güncel

Hata ve Uyarılar

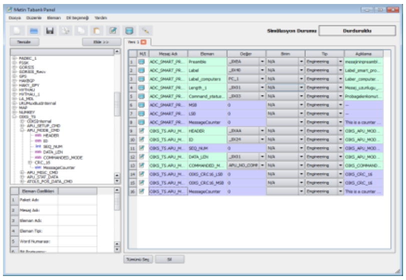
```
08:08:2014 15:47:18.5 W: system.xml acildi

08:08:2014 15:47:18.5 I: system.xml okundu

08:08:2014 15:47:18.5 I: UpdateHostDataList begin at 172.17.242.12
08:08:2014 15:47:18.5 I: Host: 172.17.242.44
08:08:2014 15:47:18.5 I: UpdateHostDataList end.
08:08:2014 15:47:18.5 I: Master
08:08:2014 15:47:20.2 I: Master
08:08:2014 15:49:47.3 I: OnDead: peer on 172.17.242.12:33664 (43680528-1f13-11e4-b1fc-d4bed994a379)
08:08:2014 15:50:41.7 I: OnDead: peer on 172.17.242.12:33663 (3f5b2f78-1f13-11e4-b6ce-d4bed994a379)
Simülasyon başlatıldı.
08:08:2014 15:51:35.3 I: Master
```

Fig. 3 SELSIM

Model User Interface (MUI): SDYA includes user interface to allow simulator runtime behavior to be monitored and for data injection. The MUI can display the status of the simulated model (i.e. a spacecraft avionic) including all messages. The simulation environment displays error messages for all erroneous conditions. Model User Interface is shown in Fig.4.



Simülasyon Durumu: Durduruldu

	M/T	Mesaj Adı	Eleman	Değer	Birim	Tip	Açıklama
1		ADC_SMART_PR...	Preamble	_0XBA	N/A	Engineering	mesajininpreambl...
2		ADC_SMART_PR...	Label	_0X40	N/A	Engineering	Label_smart_pro...
3		ADC_SMART_PR...	Label_computers	PC_1	N/A	Engineering	Label_computer...
4		ADC_SMART_PR...	Length_1	_0X01	N/A	Engineering	Mesaj_uzunlugu_...
5		ADC_SMART_PR...	Command_status...	_0X03	N/A	Engineering	Probagelenkomut...
6		ADC_SMART_PR...	MSB	0	N/A	Engineering	--
7		ADC_SMART_PR...	LSB	0	N/A	Engineering	--
8		ADC_SMART_PR...	MessageCounter	0		Engineering	This is a counter ...
9		OBKS_TS.APU_M...	HEADER	_0XAA	N/A	Engineering	OBKS_APU_MOD...
10		OBKS_TS.APU_M...	ID	_0X24	N/A	Engineering	OBKS_APU_MOD...
11		OBKS_TS.APU_M...	SEQ_NUM	0	N/A	Engineering	OBKS_APU_MOD...
12		OBKS_TS.APU_M...	DATA_LEN	_0X01	N/A	Engineering	OBKS_APU_MOD...
13		OBKS_TS.APU_M...	COMMANDED_M...	APU_NO_COM	N/A	Engineering	OBKS_COMMAND...
14		OBKS_TS.APU_M...	OBKS_CRC16_LSB	0	N/A	Engineering	OBKS_CRC_16
15		OBKS_TS.APU_M...	OBKS_CRC16_MSB	0	N/A	Engineering	OBKS_CRC_16
16		OBKS_TS.APU_M...	MessageCounter	0		Engineering	This is a counter ...

```
PADEC_1
PSSK
GORSIS
GORSIS_Reciv
GPS
HAKBGP
HAKY_GPY
HYTHAU
HYTHAU_1
LA_MDL
LRUMuxBusInternal
MAP
NUMKEY
OBKS_TS
  OBKSInternal
  APU_SETUP_CMD
  APU_MODE_CMD
    HEADER
    ID
    SEQ_NUM
    DATA_LEN
    COMMANDED_MODE
    CRC_16
    MessageCounter
  APU_MISC_CMD
  APU_STAT_DATA
  ATOLS_POS_DATA_CMD
```

	Eleman Özellikleri	
1	Paket Adı:	
2	Mesaj Adı:	
3	Eleman Adı:	
4	Eleman Tipi:	
5	Word Numarası:	
6	Bit Pozisyonu:	

Fig. 4 Model User Interface

Data Recording and Analysis Tool: Allows storing of model info and related messages to the file system on disk.

Automated Testing Tool: Provides simulation Application Programming Interface (API) to the outside world. The API is a set of functions that the application developer uses to access the features and capabilities of the system and underlying library. It supports Python, C#, C++ programming languages. With Python API, it allows value injection and monitoring of element that is used in simulation model.

Template-Based Output Generator: Generates code and build files with a given template. These outputs can be used by SELSIM, Model User Interface and Automated Testing Tool.

Node Manager: Monitors other simulation tools (SELSIM, Model User Interface so on) status, collects health info which is basically a CPU and memory usage of simulation modules from other node managers, and sends open / close or master / slave commands to nodes.

SDYA ARCHITECTURE

Development

The implementation of the architecture presented applies layered approach as compliance to SMP-2 standard. Kernel is the main component of the architecture. It includes the Common Services, Message Abstraction Layer, and Hardware Abstraction Layer (HAL). HAL is a virtualized machine that represents the true underlying hardware. HAL is provided to add avionics interfaces such as serial, MIL-STD-1553B, Ethernet, and etc. Model Manager supports simulation and model control mechanism. Message Manager allows scheduling of messages which is included in the simulation model. Data Manager supports raw to engineering conversions. Model View Abstraction Layer is based on Model-View-Controller design paradigm. SDYA Layered Architecture is shown Fig.5.

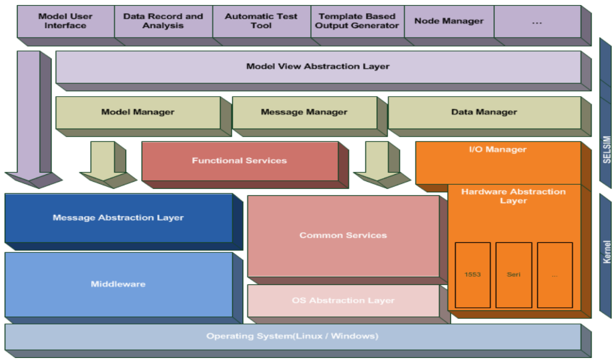


Fig. 5 SDYA Layered Architecture

SDYA serves as a real-time simulation software, which has main capabilities like interfacing with equipment, dynamic control, and data injection. In order to meet performance and scaling requirements for new models addition, a middleware that provides a messaging framework distributing models information to different nodes is used. This way, a simple, reliable, fast, and cost-efficient way of messaging between applications on various operating systems is achieved.

Distributed systems can be composed of more than one independent platform, although they may look as a single system to the user [4]. The messaging between distributed nodes requires different kind of interactions for different platforms. Publish-Subscribe mechanism has been popular among messaging patterns in recent years [5]. Publish-Subscribe meets most of the platforms needs, however, it may not provide enough flexibility for fulfilling different kind of interactions of advanced requirements of space platforms [6], such as sending, sending-confirmation, request-response, call-confirmation-response, and, progress-confirmation-update-response. For this reason, the middleware solution of SDYA is identified as ZeroMQ based on its support to different kind of messaging patterns while providing enough flexibility. An example deployment view of SDYA is shown in Fig.6.

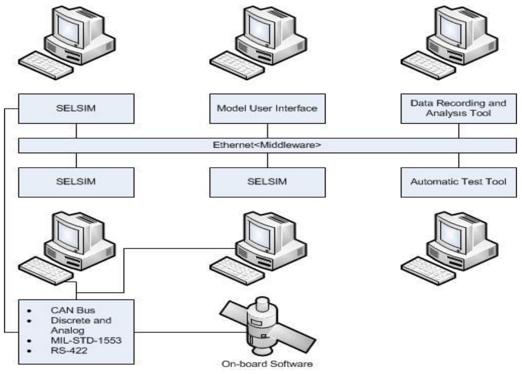


Fig. 6 SDYA Deployment View

Testing and Integration

During SDYA development, TAI processes that are compatible with CMMI Level-3 are fully applied. Independent Quality Audits are conducted for product and process compliance to requirements, and processes. For the critical/reliable flight systems, the qualification process [7] is vital. Consequently, for the qualification of SDYA, “Tool Qualification”[8][9] processes of DO-178B, which is globally accepted avionic safety-critical software development guidance, and the compatibility with ECSS-E-ST-40C[10], which is the software development standard for space engineering software developed by ESA are followed.

Moreover, “Run-time Infrastructure (Profile 2)” [11] compliance is supported by SDYA in order to conform model compatibility with ESA SMP-2 (Simulation Model Portability) standard, which is used in Satellite and Space projects. “ANKA” (unmanned aerial vehicle) is selected as the first platform application that SDYA is used. The development and adaptation of specific “ANKA” models are completed successfully.

During verification and validation process of SDYA, following activities are performed by TAI Software Development Team and TAI Independent Software Verification Team.

- Requirement, Design, Source Code and Test Case Peer Reviews in accordance with developed standards, and checklists
- Unit Tests
- Requirement-Based Tests
- Software Integration Tests

Requirement-Based Tests are developed and conducted as automated, semi-automated and manual tests. For efficiency and repeatability, automatic test design, and execution is encouraged, and thus maximized. In respect to this goal, test environments which are compatible with automatic test run operations are developed. This testing infrastructure helped us in detection and resolution of errors in initial development, integration phases with the help of continuous (nightly) test runs. For the verification of Graphical User Interfaces (GUI), test cases are developed by using a third party verification tool supporting QT environment. Python script language is selected as the test development language. All tests are performed on Windows and Linux operating systems for satisfying the cross platform objective.

SPACE DOMAIN UTILIZATION OF SDYA

The simulation engine of the SDYA provides all the required test facilities, to support verification at unit level and subsystem level. Simulator software is structured in models and avionic buses. Thus, it contains sensor models, actuator models, and etc.. Models can implement a complete simulation for a certain unit, or just the required signals to supply the unit hardware in the loop. It is able to run in near real time, and provides visualization and logging of all interested data. Its architecture is modular and object-oriented. With these basic capabilities, and its scalable, modular, and distributed architecture TAI is planning to utilize SDYA as:

- A Validation tool for the :
  - Performance and Robustness Verification,
  - Software Verification Facility (SVF),
  - Avionic Functional Chain Validation (FCV),
  - Satellite Assembly, Integration and Test (AIT) procedures.

- A support tool for the:
  - Validation of Spacecraft operational procedures,
  - Validation of Satellite Control Centre,
  - Spacecraft operators training,
  - Analyze and/or investigation of the anomalies detected in flight.

CONCLUSIONS

This paper presents SDYA architecture, its generic simulation infrastructure and development process. The use of SDYA in the validation process of Satellite Projects is planned and it will be the basis for a Simulator Software Product Line to be formed at TAI. With its current capabilities, and flexible architecture, SDYA will support a large range of Satellite integration and test activities:

- Software Verification Facility (SVF)
- Functional Chain Validation
- Avionic Test Bench Simulation
- Dynamic Satellite Simulation (DSS)

SDYA gives the test and integration teams a powerful and flexible tool that de-risks the qualification process and improves its efficiency. Nevertheless, it is clear that the testing and validation process of Spacecraft Simulators can be improved with the introduction of new testing techniques and procedures, such as increased usage of automated tests. SDYA-based systems could be developed to support integration and testing of complex systems in space domain.

REFERENCES

[1] Nancy G. Leveson, A New Approach To System Safety Engineering, Aeronautics and Astronautics Massachusetts Institute of Technology, June 2002.
[2] Jens Eickhoff ,Simulating Spacecraft Systems, Springer Series in Aerospace Technology,2009
[3] SEI, “CMMI for Development”, Technical Report, CMU/SEI- 2010-TR-033, Software Engineering Institute, Carnegie Mellon University, 2010.
[4] Kim, H-D. 2001. “An XML-based modeling language for the open interchange of decision models,” Decision Support Systems 31, Issue: 4 (Oct), 429-441.
[5] Tanenbaum, van Steen, “Distributed Systems, Principles and Paradigms”, Prentice Hall, 2002.
[6] IBM, “WebSphere MQ Publish/Subscribe”, http://bitly.com/Q8h6KB, 2009.
[7] “Mission Operations Message Abstraction Layer. Recommendation for Space Data System Standards”, CCSDS 521.0-B-1, Blue Book. Issue 1, Washington, D.C.: CCSDS, October 2010

[8] RTCA DO-178B/ED-12B Section 12.2
[9] ECSS,ECSS-Q-ST-80C,“Space Product Assurance/Software product assurance”, ESA-ESTEC, 6 March 2009
[10] ECSS,ECSS-E-ST-40C,”Space Engineering /Software”, ESA-ESTEC, 6 March 2009
[11] Simulation modelling platform - Volume 1: Principles and requirements, ECSS-E-TM-40-07 Volume 1A, 25 January 2011