AVES SDK: Bridging the Gap between Simulator and Flight Systems Designer

Torsten Gerlach[1] and Umut Durak[2]
German Aerospace Center (DLR), Institute of Flight Systems, Braunschweig, 38108, Germany

Developing and integrating a new concept in a flight simulator usually requires rigorous implementation and integration steps which involves various parties. German Aerospace Center (DLR) is developing and operating the Air Vehicle Simulator (AVES), and focusses on shortening the simulator implementation and integration lead time. This effort targets at bridging this gap by promoting the software infrastructure of the flight simulator as AVES Software Development Kit (SDK) for flight system designers and simulator end users.

Nomenclature

ACT/FHS	=	Active Control Technology/Flying Helicopter Simulator	ICAO	=	International Civil Aviation Organization
API	=	Application Programming Interface	IOS	=	Instructor Operator Station
ARINC	=	Aeronautical Radio Incorporated	MilBus	=	military bus system, MIL-STD-1553
ATRA	=	Advanced Technology Research Aircraft	QNX	=	unix based real-time operating system
ATTAS	=	Advanced Technologies Testing Aircraft System	SCM	=	Source Code Management
			SDK	=	Software Development Kit
AVES	=	Air Vehicle Simulator	SVN	=	Subversion
CAN	=	Controller Area Network	SWTD	=	Software Test Device
DMC	=	Display Management Computer	TCP	=	Transmission Control Protocol
HLA	=	High Level Architecture	UDP	=	User Datagram Protocol
IC	=	Interface Computer	XML	=	Extended Markup Language

I. Introduction

THE aeronautics community has been utilizing flight simulators as computational testbeds for flight systems development and human factor studies for 30 years. As virtual test beds, flight simulators are employed to evaluate concepts, conduct Pilot-in-the-Loop experiments and collect valuable user experience data. Furthermore, they are extensively promoted to reduce the number of flight tests. Various aircraft manufacturers, as well as research organizations, have built their own research and engineering flight simulator facilities. Some of the well-known early examples are the ATTAS Ground Based Simulator from the German Aerospace Center (DLR),[1,2] NASA Crew Vehicle Systems Research Facility in Ames Research Center[3] and Visual Motion Simulation and Cockpit Motion Facility of NASA Langley Research Center.[4] Latest facilities that are in use include the Air Vehicle Simulator (AVES) of DLR,[5] HELIFLIGHT from the University of Liverpool,[6] NASA Ames Vertical Motion Simulator,[7] SIMONA at Delft University of Technology[8] and F-35 engineering simulators[9] in Fort Worth, Texas.

The DLR Institute of Flight Systems (FT) is involved in developing and employing simulators in flight systems studies for a long time. AVES can be pronounced as the third technology generation research simulator after analogue computed ground based simulator for DLRs in-flight simulator HFB320[10] in the 1970`s and the ATTAS Ground Based Simulator and ACT/FHS EC135 Ground Based Simulator facilities[11] from 1980`s. The new setup focusses on a highly flexible, beyond state of the art research flight simulation facility. Its development started in 2009 and its full operation capability was reached in June 2013. AVES consists of two simulators with a Roll-on/Roll-off (RO-RO) cockpit exchange concept. A full motion based and a fixed based simulator can carry arbitrary cockpits and interchange them within a few hours. Currently two cockpit infrastructures are available: an Airbus

[1] Group Leader, Flight Dynamics and Simulation, torsten.gerlach@dlr.de
[2] Research Scientist, Flight Dynamics and Simulation, umut.durak@dlr.de, AIAA Member

A320 and an Airbus Helicopters EC135. These cockpits correspond to DLR's flying testbeds, A320 ATRA[12] and ACT/FHS EC135.[13,14]

Decades of experience working with flight simulators indicated that there exists a gap between flight simulator and flight systems designers. Introducing and testing a new concept in a flight simulator can usually be achieved after a burdensome implementation and several integration steps which involve various parties. However, recently agility becomes a desired quality of the systems development process. In iterative design cycles different options need to be rapidly evaluated in simulation studies. State-of-the-Art Model Based Systems Development practice promotes extensive Model-in-the-Loop, Software-in-the-Loop and Hardware-in-the-Loop testing, thus streamlines the implementation of the design to a real product.

DLR FT is targeting at bridging this gap by promoting a flexible and adaptable simulation software suite, AVES Software Development Kit (SDK). AVES SDK can be used in all Model-in-the-Loop, Software-in-the-Loop, Simulator-in-the-Loop and Simulator modes. The Model-in-the-Loop mode will support the flight systems researcher to integrate and test flight system models in the desktop environment. The Software-in-the-Loop mode will be used to integrate and test the flight system software in the real-time Software Test Device (SWTD) environment that has almost all the hardware and software elements of AVES in a systems integration laboratory setting. The Simulator-in-the-Loop model will enable to integrate and test flight system models or software with the simulator utilizing the developer station in the simulator control room. These modes provide the users of the AVES capabilities to mitigate integration risks of their models and software early in the desktop simulation step. Real-time requirements can be checked in SWTD without any requirement to simulator hours. The Simulator-in-the-Loop mode additionally supports the simulator debugging and enables an agile simulator time error correction or a feature update.

Developing flexible and adaptable flight simulation first got some attention in the 1990's. In 1994, as Delft University of Technology was building their research simulator SIMONA,[15] they aim at obtaining a highly flexible system to support various research challenges. They report that it is only achievable by optimized hardware/software and advanced software development techniques. It is presented that SIMONA is built on off-the-shelf PC based hardware platform. This enabled the portability of the software to the other different PC based hardware platforms and utilization of the advanced and high quality integrated development environments. The SIMONA team built their software development on the idea of spiral lifecycle model, supported with metaprogramming and automated code generation. Thus, it has been possible for them to boost flexibility and upgradeability via code level reuse. The details of the SIMONA software architecture, metaprogramming and code generation were introduced by van Gool in 1996.[16] While SIMONA effort targeted at a configurable and flexible simulator, all the capability was planned and developed for the simulator facility. The novel approaches to simulator development surely decreased the simulator integration time for the researcher and inspired the following research simulator development efforts, like AVES. However, they had no intention to make the simulator available at researcher`s desktop.

The Avionics Integration Research SIMulator (AIRSIM)[17] was developed by The National Aerospace Laboratory (NLR) in Netherlands as a low cost desktop flight simulator in 1998. Its development philosophy was to provide developers with a high fidelity simulation capability at their desktop without the use of an entire simulator facility. They used the software from the simulator facility to construct a desktop simulator configuration. While it was the early example that implemented the idea to provide a desktop version of the simulator to the researcher, the configurability and flexibility qualities were not focused.

In 2000, Ippolito and Pritchett[18] present their architecture for a configurable flight simulator. The architecture introduces an interface standard among simulation components and provides a mechanism to register, initialize and manage the data traffic among the components. The component architecture is based on plug-in strategy and networking is supported over High Level Architecture. The configurability and flexibility is enabled via plug-ins that may also be swapped during run time. Object oriented principles are employed over the component interfaces to enable message and data passing.

In 2008, Landry[9] from Lockheed Martin Aeronautics Company presented how they reused flight simulation components like aircraft trim, linearization of the simulation model and a virtual cockpit through the various targets including desktops, engineering labs, operational analysis labs as well as fixed and motion based flight simulators throughout the development lifecycle of the F-35 fighter aircraft. He reports an inspiring case how the simulator software components can be used during development life cycle in various targets, but the manuscript does not focus on explaining the technical solution underneath.

II. AVES

A. Simulator Design

AVES was designed regarding the best practices of the simulation industry that have been compiled as certification regulations for flight training devices like the ICAO 9625 Manual Criteria for the Qualification of Flight Training Devices.[19] As presented in Fig. 1, the motion simulator consists of a six degree of freedom stewart platform with high fidelity, a large 15 channel front projection system with 240 degrees horizontal and up to 95 degrees vertical field of view. To provide the same environment in the fixed-based installation, this simulator also uses a 15 channel projection system with the same specifications. Both simulation cockpits are equipped with replicas of the real cockpit devices. Controls are simulated using active control loading systems, which can be tuned according to any aircraft specification or research requirements. The helicopter cockpit uses an eight channel control loading system, whereas the A320 uses active sidesticks instead of passive ones. This enables AVES to support future research in active control technologies in modern aircraft.[20] The real cockpit environment is supported by large operator cabins, to control the simulation, observe the simulator trial or develop software right in the place. The software infrastructure is based upon an open system architecture strategy facilitated through employing industry standard programming and modeling languages and operating systems on state-of-the-art commercial off the shelf (COTS) hardware platforms.

The main areas of employment can be introduced as research in new flight system concepts, testing and demonstrating new aircraft concepts, research in simulation and training technology and flight test preparation. AVES is not planned as a certified training device. Rather the capability to alter and configure almost every aspect of the flight simulator in a short amount of time for supporting diverse research requirements has been targeted.

Figure 1. Air Vehicle Simulator - AVES.

B. Software Infrastructure

AVES utilizes an open architecture strategy. Components can be swapped, added and updated easily. The main focus is on a distributed simulation. Components are designed to be running as separate processes in a loosely coupled fashion on the same or separate hardware. As Allerton[21] promotes, the components of a flight simulation are mostly structured around the equations of motion. But the architectural design of AVES follows a different perspective. All simulator components including the flight dynamics model and the flight system simulations are connected to a centralized node, which is running under real-time conditions. Figure 2 depicts this topology.

The simulation system is designed in a star topology. The central component is called Interface Computer (IC), and can be pronounced as the core process, which deals with data distribution, computing the flight systems simulation and connecting all simulation parts together. The networking of the hardware and the software components is done in various ways. Most of the data exchange is carried out via Ethernet using UDP and TCP. The Cockpit hardware is connected via CAN-Bus, ARINC or MilBus. All software modules run on AVES target computers which have either Microsoft Windows operating system for soft real-time components or QNX for hard real-time components. To boost the reuse of software and minimize the effort in developing real-time distributed simulation that utilizes various networking protocols an overall simulation framework, *2Simulate*[22] has been developed. It is delivered in three parts, which cover all aspects of a real-time simulation application.

2SimRT is the library used to create applications in hard- (QNX) or soft (Windows) real-time environments. It delivers a large set of tasks that can be used to connect other simulation parts, to run simulation models and it pro-

vides a data dictionary, accessible from outside. Its programming interface enables a rapid development cycle without expert knowledge in real-time programming.

2SimMC is the generic interface for the model integration library. It is connected natively to *2SimRT* and enables the integration of models that have been developed using various types of modeling tools and languages, e.g. MATLAB/Simulink. *2SimMC* provides a data dictionary to have an easy access to the model interface signals, internal state variables and parameters. A unified model integration process delivers rules for modeling aspects and interfaces.[23] The outcome is an executable model either in hard or soft real-time environments.

2SimCC is the graphical user interface for a 2Simulate infrastructure. It connects so called *targets*, which are other *2SimRT* applications. Once connected *2SimCC* can control the targets, offers access to the data dictionaries with visualization possibilities like plots. Furthermore, the user can create own panels and select from various controls to display connected signals in the data dictionary.

2Simulate as the underlying infrastructure covers a large feature set for the most important simulation components, the flight model, the interface computer and the instructor operator station. A detailed look reveals a variety of other important applications connected and listed in the following.

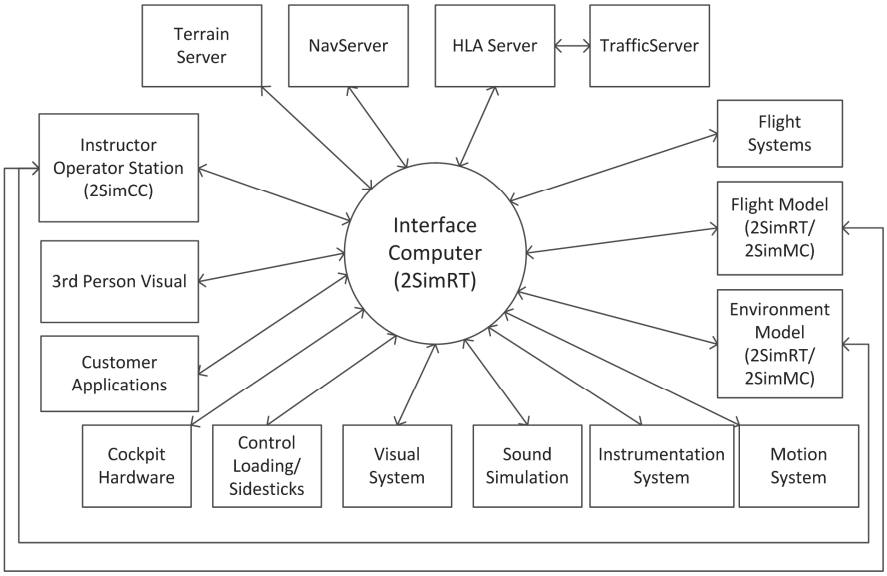

Figure 2. Main Parts of AVES Simulation Infrastructure.

Interface Computer

This *2SimRT* based application distributes all simulation data and is equipped with the data processing routines for manipulating data for particular usage. It is aircraft type specific software fitted to the AVES cockpit hardware. The IC can be configured via XML for network connections and automatic data processing.

Flight Dynamics Model

The main part of a flight simulation is the flight dynamics model. Development can be done in several modeling languages. AVES supports MATLAB/Simulink and C/C++ models. The simulation integration is done with using *2SimMC* and creating a *2SimRT* application.

Instructor Operator Station

The IOS is a customized application using *2SimCC* with adding an aircraft type specific functionality. It controls the overall simulation. The data dictionary functionality is able to show and modify any simulation parameter inside the connected target applications.

Aircraft System Simulation Software

Simulating aircraft systems is one of the most important parts in flight simulator development. It ensures the correct functionality of the aircraft during operation. System simulation components can be used as external standalone software modules or, if platform independent can be integrated to the IC.

Instrumentation System

Modern aircraft are equipped with digital displays to present the most important information during flight. AVES uses the in house product 2Indicate[24] to generate the graphics for A320 and EC135 displays. As 2Indicate is platform independent, the display systems can run under Windows or Linux. Displays are interpreted and not need to be compiled or linked. The logical computation is done by a separate aircraft specific system software, e.g. for an emulation of the A320 Display Management Computer (DMC).

Visual System

A simulator visual system consists of a rendering hardware, e.g. COTS gaming PC, an engine for rendering the image and a 3D database for terrain visualization. Because of fast changing requirements on visual simulation and database generation, DLR developed its in house AVESImageGenerator. It is an OpenGL based rendering system using OpenSceneGraph[25] as API.

Other Peripheral Applications

Beneath the main modules of AVES flight simulation, there are a lot of other necessary programs to complete the simulation. The sound simulation is connected via Ethernet and is responsible for aircraft specific noise and air traffic communication. A connection to a High Level Architecture (HLA) server enables the simulator to connect to other participants, like the traffic server for simulating generic air and ground vehicles.

III. User and Developer Requirements

Operating and developing a flight simulator with an open architecture infrastructure requires a well-defined set of rules to develop and maintain software. The users of the simulator focus on software reuse from simulation architecture as much as possible and they want to integrate their code or models as quick as possible. On the other hand, the developers of the simulator try to keep it stable and maintainable. The simulator developer objectives usually collide with the needs of the simulator user. The particular requirements and conflicting perspectives are provided below.

Simulator Developer perspective

- Full authority over software source code with fast access to third party software products
- Source code management
- Configuration management
- Bug and feature tracking procedures
- Deployment management
- Short release cycle
- Test strategy

Simulator User perspective

- Easy access to the simulator and the simulation infrastructure
- Fast integration of flight dynamics model or system simulation software
- Autonomous work and supported work
- Bug and feature tracking with short solution time
- Simulation use only or developing software to be integrated
- One click solutions

Two different main use cases can be identified for the simulator. The first one is simulator use only, which concerns the use of the unmodified simulation infrastructure for experiment. The second one is for the research purposes, where code developed by the user or new models shall be integrated. These artifacts do often not comply with the infrastructure and must be modified to be integrated. This has to be done by the simulator developers, thus requires coordination and planning and causes a time delay.

As a research facility, rapid prototyping and agile model based development practices are the favorable strategies for software development. The simulator user shall be welcomed as a part of the development team, with a supported development kit.

Managing a process with multitude of teams with conflicting perspectives requires a clearly defined business processes. Figure 3 describes the procedure to conduct a new experiment in general. A contact person from the AVES team is chosen at the beginning. As airspace projects have a long term research perspective, it is necessary to keep a tight relation between the simulator team and the user. The contact person coordinates the integration and implementation activities. A general and specific introduction to the simulator is given before full usage of the facility can be permitted. After setting up the environment, the first simulator runs can be conducted. If bugs have to be solved or new features have to be implemented, the development stage is reached again. A successful experiment or project ends up with a lessons learned session.

This flexible way of using AVES requires a fast development cycle with the involvement of the simulator users. On the other hand, it comes with a set of technical requirements for the simulator software components which are listed below:

Rules for simulation software

- Running in several execution environments (Desktop, Test Device, Simulator)
- Emulation for unavailable hardware devices
- Configurable standalone software
- Platform independency
- Source code management and easy access
- Automated processing of interface changes

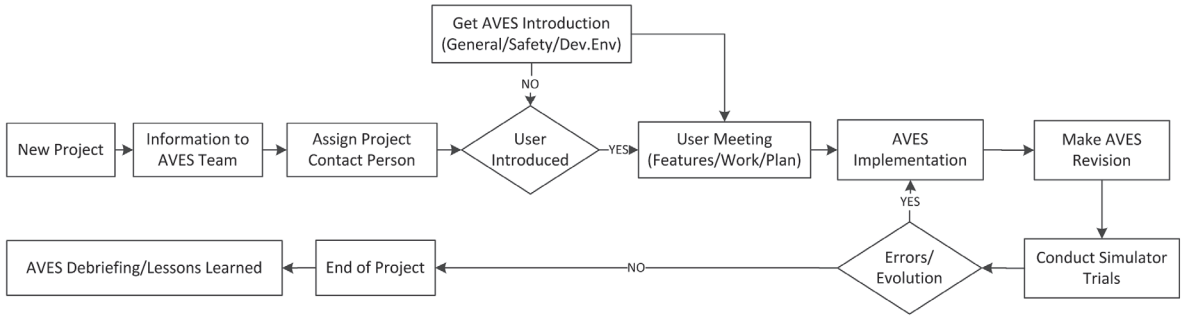

Figure 3. Conducting a simulator campaign in AVES.

IV. AVES Software Development Kit

AVES SDK bridges the gap between the simulator user and the developer. It is a set of development rules, automatic scripting files and a general organization structure for development and runtime. The main idea of reducing time to simulation with involving the user and unifying the simulator software setup leads to four different areas of applications. Figure 4 presents these areas as desktop, software test device, developer station and AVES simulator.

To enable the workflow from desktop development to simulator deployment, AVES SDK provides distinct configurations for particular steps or stages. The Model-in-the-Loop (MIL) and Software-in-the-Loop (SIL) configurations are for the early rapid prototyping stage. The support incorporates a software installation on the engineers desktop. It is a full set of simulator applications ready to use. Alternatively it is possible to get support from a so called Software Test Device (SWTD). SWTD is a MIL/SIL environment that utilizes similar hardware and operating system configuration with the simulator, thus it is a real-time software integration lab for the simulator. It is located in the simulator building.

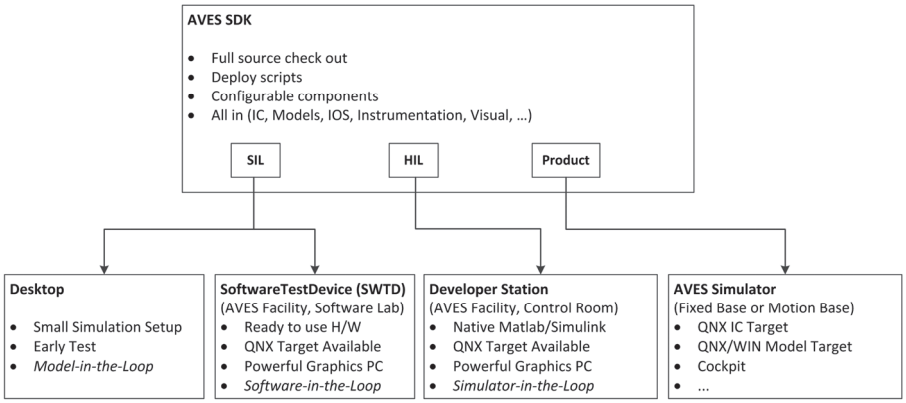

Figure 4. AVES SDK – Development Cases and Target Platforms.

The next stage after a successful software development and conducting test and integration on desktop or if necessary on SWTD, is the Hardware-in-the-Loop (HIL) environment. A developer station is provided in the simulator control room which is equipped with powerful hardware and a QNX target PC. It can directly be connected to the real flight simulator. This development state is called Simulator-in-the-Loop. It enables a precise test and debugging. The successful test leads to the deployment of the newly developed simulator component, which is a new software version for AVES. If

this version is tested with the full flight simulation, a new software release is made, and a new simulator version can be established.

The AVES SDK infrastructure is constructed in a way, such that it distinguishes between runtime and development. The Runtime Environment (RTE) is the complete set of ready to use simulator software products. They must be configured and started from one location. Even deployment to the simulator shall take place here. The development can take place in a different area, which is more or less free to the developer. This Software Development Environment (SDE) is tightly connected to the runtime and defines a set of rules to create own software.

As AVES is currently equipped with two different hardware infrastructures, which are related to the A320 and EC135 cockpit, the SDK has to distinguish between them. Furthermore, the interchange possibility introduces a second level of complexity, which is reflected in the different configuration modes. As everything shall be easy to be installed from desktop to simulator hardware, one infrastructure was created (Fig. 5).

With using a Source Code Management (SCM) system, it is possible to automate various functions of the configuration management. Automatic deployment is one of these functions. Starting from the root folder *AVESSDK*, which can be checked out from the SCM system, four levels of the infrastructure setup are applied. The setup separates software environments as runtime and development. The AVES simulation hardware specific folders reflect the aircraft type. Future simulation cockpits can be easily integrated to this structure. Additionally a general purpose software development area is available in SDE for basically general purpose tools and utilities like AVESImageGenerator or 2Simulate core software.

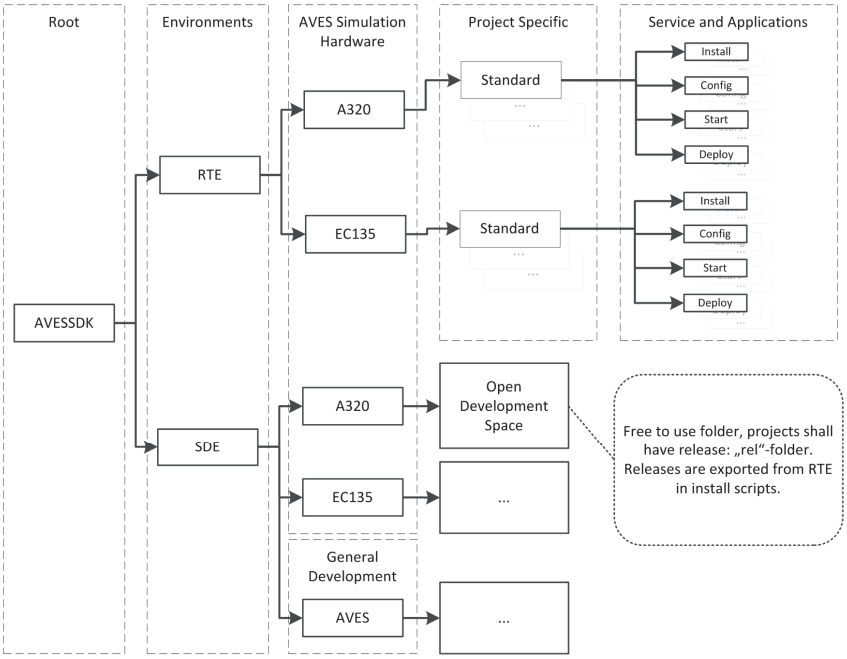

Figure 5. AVES SDK organization structure.

With respect to different projects, RTE subfolders contain particular simulator components. The Standard project is the main execution environment for A320 and EC135 aircraft type. This baseline project composes the complete simulator components. Different projects may have their own project area and use project specific components from this folder. For example, developing a new display system with an interface to the standard IC for A320. The project specific area makes use of four standardized scripting and configuration sub-structures inside the service and application level.

- *Install* includes a set of batch files to export the current version of the simulator software from the SCM system. The products are installed into their own folders.
- *Config* holds all configuration files. AVES makes strong use of XML for configuring the applications, like IP addresses or port numbers.
- *Start* is the main folder for running all software. For the real simulator, this is the place to bring life to the simulation.
- *Deploy* is responsible for distributing the installed software parts to its target hardware. It uses automatic network connect, copy and disconnect procedures.

As mentioned before, the AVES cockpit exchange concept introduces at least two levels of different configurations. The motion and fixed-base possess distinct sets of hardware, thereby they require a decoupled configuration set. *Install*, *Config*, *Start* and *Deploy* are subdivided into a *Motion* and *Fixed* subfolder. Because of different network addresses and hardware setups, they contain specific XML and batch scripts. To support non-simulator hardware

utilization, additional sub-folders can be introduced. For the desktop version, a third one, *Desktop* is used. All modules are configured to exchange their data on a local host.

But controlling and maintaining the software is only one part. The link to the SDE is done with three directories inside each installed application. The *current* application is the latest version with a revision from the SCM. It is exported using the install script and is tested, stable and fully configured. Software inside *dev* is still under development, it can be exported from SCM or be directly copied from the SDE. As the complete infrastructure is related to the root *AVESSDK*, it is easy to create scripts for automatic deployment to *dev*, inside the software project with relative paths. As a service area, the *rev* subfolder keeps old stable versions as a copy on the target. It might be necessary to switch back to a previous version for testing purposes, for example, if current software is faulty. A full RTE installation includes all configured applications to run the simulation either on desktop or in the flight simulator. With making use of relative file paths, no script or batch file need to be changed.

The source for RTE components is the SDE area. It is a free development space, but also structured regarding specific cockpit configurations. The projects are checked out from the SCM into their specific folders. A project specific substructure as in RTE is possible but not necessary. General development can take place in the *AVES* folder.

All software projects shall have a special release folder. It needs to contain a full set of the application to run it in a standalone manner. This includes all executables, libraries, configuration files or data files for execution. After successful test, a revision of the development project is made. This revision is the basis for RTE's install scripts.

V. Use Case: Update AVES A320 Flight Management System

The AVES A320 Flight Management System (FMS) is a software simulation that composes several aircraft systems together. It contains the Flight Management and Guidance System (FMGC), the Multi Purpose Control and Display Unit (MCDU) and the corresponding performance calculation for flight planning. All parts are separate software projects developed by aircraft systems specialists and are therefore stand-alone projects. To make use of hard real-time processing and enable a simpler debugging in the simulator, all three software projects are linked together in one FMS library. The AVES interface computer for A320 hosts them as a separate task. This module mainly needs inputs from the A320 cockpits Flight Control Unit (FCU), the Electronic Flight Instrumentation System (EFIS) and the MCDU device itself.

Figure 6 describes the workflow for the system developer from desktop to the A320 simulator. Standalone development of the modules takes place inside the SDE. After successful implementation, the IC project exports its sources to an external project folder.

This is linked to the main source of the IC, and generates an executable image. The most difficult step is aligning the interface definitions between FMS integrated modules and the Interface Computer. To minimize error and effort, an A320 signal database was set up. It is capable of exporting ready to use header and definition files.

A FMS desktop test environment is easily applied from RTE`s available applications. It contains the main simulation parts, e.g. flight model, instrumentation system and system simulation.

As a real cockpit is not available, a cockpit emulation as part of the IOS is used. Control inputs are generated with a joystick.

A successful test leads to committing

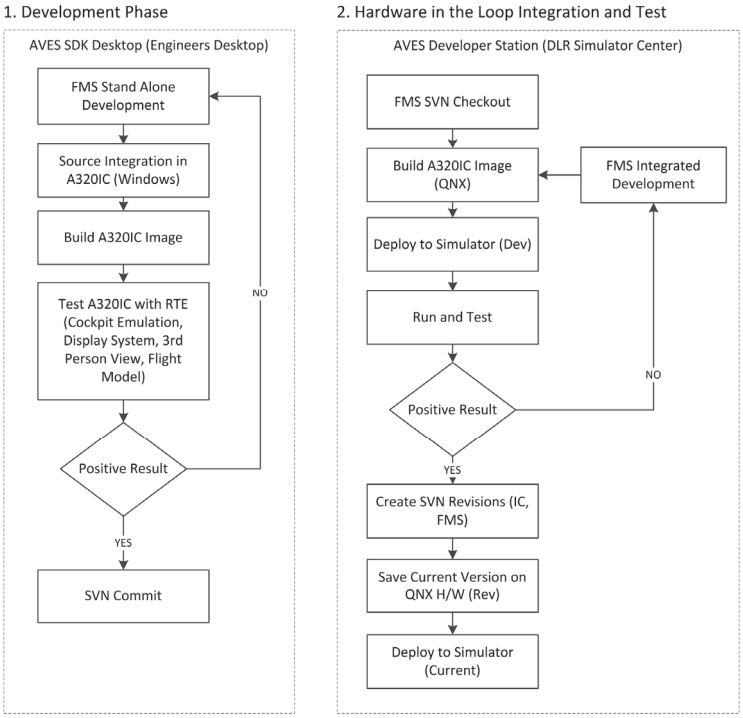

Figure 6. FMS development and Integration workflow.

the source files to the SCM. The Developer Station at AVES simulator facility is the next step. Due to the already tested version of the Windows A320 IC, it is only necessary to create a QNX image.

The similar infrastructure at the engineers' desktop and at the simulator facility enables the developer to quickly build the application after checking out the new source code. Through the Simulator-in-the-Loop capability the new IC can be tested in its native environment. If QNX behavior is different, an online debugging in hard real-time can be applied. A positive test leads to a new SVN revision of the interface computer and, if modifications have been done, of the FMS. The new revision is deployed to the current folder under a new version number.

VI. Conclusion

The paper presents the AVES Software Development Kit that has been developed in order to bridge the gap between the Simulator User and the Simulator. It provides a reconfigurable collection of simulator software components that may be deployed on the simulator users' desktop, the software test device, the developer station in the simulator control room and the simulator itself. Thus, the Simulator User is enabled to integrate and evaluate new code or model in a Model-in-the-Loop, Software-in-the-Loop and Simulator-in-the-Loop fashion. This capability not only provides the unique opportunity to the simulator user to make early tests and mitigate any simulator integration risks as early as possible, it also provides a test bed in on the desktop so that he can rapidly prototype various concepts and conduct early evaluation without requiring the complete simulator facility.

The technical background of the AVES SDK heavily relies of source control, build and deployment automation utilizing command line scripting. Further deployment on various targets is enabled with strictly sticking to portable coding practices.

This paper also presents an application case study to illustrate how AVES SDK can be utilized within a process such that the simulator software and model development is streamlined as much as possible. One of the most complex components of an A320 simulation, the Flight Management System is picked as the show case. Thereby, the reader is provided with a rather representative example.

While AVES SDK is functioning to a great extent with scripts and configuration files, as the next step, it is planned to develop a model based simulator configuration and deployment tool. With this tool it is envisioned to model any deployment scenario, like desktop or Software Test Device as a model. Then, via model-to-text transformations, all scripts and the configuration files are generated automatically.

References

[1]Saager, P., "Real-Time Hardware-in-the-Loop Simulation for 'ATTAS' and 'ATTHeS' Advanced Technology Flight Test Vehicles," *AGARD Guidance and Control Panel*, 50th Symposium, Izmir, Turkey, 1990.

[2]Klaes, S., "ATTAS Ground Based System Simulator -An Update-," *AIAA Modeling and Simulation Technologies Conference and Exhibit*, Denver, CO, 2000.

[3]Sullivan, B. and Soukup, P., "The NASA 747-400 Flight Simulator: A National Resource for Aviation Safety Research," *AIAA Flight Simulation Technologies Conference*, San Diego, CA, 1996.

[4]Smith, R., "A Description of the Cockpit Motion Facility and the Research Flight Deck Simulator," *AIAA Modeling and Simulation Technologies Conference and Exhibit*, Denver, CO, 2000.

[5]Duda, H., Gerlach, T., Advani, S. and Potter, M., "Design of the DLR AVES Research Flight Simulator," *AIAA Modeling and Simulation Technologies (MST) Conference*, Boston, MA, 2013.

[6]White, M. and Padfield, G., "The Use of Flight Simulation for Research and Teaching in Academia," *AIAA Atmospheric Flight Mechanics Conference and Exhibit*, Keystone, CO, 2006.

[7]Advani, S., Giovannetti, D. and Blum, M., "Design of a Hexapod Motion Cueing System for NASA Ames Vertical Motion Simulator," *AIAA Modeling and Simulation Technologies Conference and Exhibit*, Monterey, CA, 2002.

[8]Stroosma, O., van Paassen, R. and Mulder, M., "Using the Simona Research Simulator for Human-Machine Interaction Research," *AIAA Modeling and Simulation Technologies Conference and Exhibit*, Austin, TX, 2003.

[9]Landry Jr, L. M. M., "Application of Modeling, Simulation and Labs to the F-35 Program," *AIAA Modeling and Simulation Technologies Conference and Exhibit*, Honolulu, HI, 2008.

[10]Hamel, P., *Fliegende Simulatoren und Technologieträger*, 1st ed.,Verlag Appelhans, E, Braunschweig, Germany, 2014.

[11]Klaes, S., "ATTAS & ACT/FHS System Simulation For Pre-Flight Software and Hardware Testing," *AIAA Modeling and Simulation Technologies Conference and Exhibit*, Monterey, CA, 2002.

[12]Schneckenburger, N., Klein, C., and Schnell, M., "OFDM based data link for the DLR research aircraft ATRA," *IEEE Integrated Communications, Navigation and Surveillance Conference (ICNS)*, Herndon, VA, 2011.

[13]Kaletka, J., Kurscheid, H., and Butter, U., "FHS, the new research helicopter: Ready for service," *Aerospace Science and Technology*, Vol. 9, No. 5, 2005, pp. 456-467.

[14]Ockier, C., and Butter, U., "ACT/FHS–an airborne rotorcraft simulator for technology development and research," *AIAA Modeling and Simulation Technologies Conference*, Denver, CO, 2000.

[15]Theunissen, E., and Lamerigts, T., "Increasing the flexibility and reducing the cost of flight simulators," *AIAA Flight Simulation Technologies Conference*, Scottsdale, AZ, 1994.
[16]van Gool, P. C. A., "Software design for a reconfigurable simulator," *AIAA Flight Simulation Technologies Conference*, San Diego, CA, 1996.
[17]Groeneweg, J., "AIRSIM, A desktop research flight simulator," *AIAA Modeling and Simulation Technologies Conference and Exhibit*, Boston, MA, 1998.
[18]Pritchett, A. R., and Ippolito, C., "Software architecture for a reconfigurable flight simulator," *AIAA Modeling and Simulation Technologies Conference*, Denver, CO, 2000.
[19]International Civil Aviation Organization, *Manual Criteria for the Qualification of Flight Training Devices*, 3th ed., ICAO, Quebec, Canada, 2009.
[20]Fonseca U. A. and Niedermeier, D. "Limited evaluation of the influence of coupled sidesticks on the pilot monitoring's awareness during stall in cruise," *AIAA Modeling and Simulation Technologies Conference*, Boston, MA, 2013.
[21]Allerton, D, *Principles of Flight Simulation*, 1st ed., John Wiley & Sons, Hoboken, NJ, 2009.
[22]Gotschlich, J., Gerlach, T. and Durak, U., „2Simulate: A Distributed Real-Time Simulation Framework," *ASIM STS/GMMS Workshop 2014*. Reutlingen, Germany, 2014.
[23]Gerlach, T., Durak, U. and Gotschlich, J., "Model Integration Workflow for Keeping Models up to Date in a Research Simulator," *Simultech 2014*, Vienna, Austria, 2014.
[24]Richter, R., "2Indicate Flexible Visualisierung technischer Prozesse," *DLR Nachrichten*, Vol. 112, 2005, pp. 50-51.
[25]Wang, R., Xuelei, Q., *OpenSceneGraph 3 CookBook*, 1st ed., Packt Publishing, Birmingham, United Kingdom, 2012.