

Interactive Design and Simulation Platform for Flight Vehicle Systems Development

Hugh H. T. Liu*

University of Toronto, Toronto, Ontario, M3H 5T6, Canada

and

Holger Berndt†

Universität Stuttgart, Stuttgart, Germany

The need for integrated systems development arises from complexity of a system. This paper presents an interactive design and simulation platform for flight vehicle systems development. Its “connect-and-play” capability and adaptability enable “on-line” interaction between design and simulation during the integrated development. As a case study, the implementation of the proposed platform and an aircraft flight control system development example are demonstrated on an experimental test bed including a real-time systems simulator and a flight training device.

I. Introduction

THE engineering systems development process typically has a waterfall view as if the different development stages were performed chronologically and independent to each other, when in reality they are extensively interrelated and interconnected. Moreover, a system often consists of multiple subsystems and components that are also interacting or even have conflicting characteristic features. Therefore, the entire development relies on iterative cycles between design modifications and integration and testing verifications, until the final design “converges.” Unfortunately, this also makes the design process time-consuming and fragile: A slight change may require a completely new cycle of redesign. The need for an integrated development process arises from complexity, such as in many engineering intensive applications from automotive and aerospace industry. A research program entitled Software Enabled Control (SEC) is initiated to integrate multi-modal, coordinated operation of subsystems, and enable large-scale distribution of control.¹ Under this program, an open control platform (OCP) is being developed to integrate control technologies and resources.^{2,3} The emerging field of multi-paradigm modeling addresses the directions of research in model abstraction, transformation, multi-formalism modeling, and meta-modeling.⁴ They are concerned with the models of system behavior, the relationship between models at different level of details, conversion of models expressed in different formalism, and the description of classes of models, respectively.

One of the key requirements for an integrated development process is the “plug-and-play” capability. It enables decoupled component (subsystem) design and integrated testing based on the same model. Therefore, design decisions are gradually finalized through iterations between component-level and system-level development. From complex control systems development point of view,⁵ advocate the computer automated multi-paradigm modeling.⁶ propose an actor oriented control system design approach, where the design acknowledges the variety of interaction models among components, and express these interaction styles independently from the functionality of components.

Received 23 January 2006; revision received 03 May 2006; accepted for publication 10 May 2006. Copyright 2006 by Hugh H. T. Liu and Holger Berndt. Published by the American Institute of Aeronautics and Astronautics, Inc., with permission. Copies of this paper may be made for personal or internal use, on condition that the copier pay the \$10.00 per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923; include the code 1542-9423/04 \$10.00 in correspondence with the CCC.

* Associate Professor, Institute for Aerospace Studies (UTIAS), 4925 Dufferin Street, AIAA Member.

† Exchange Research Intern to the UTIAS, Institut für Flugmechanik und Flugregelung.

Müller-Glaser *et al.*⁷ categorize the existing strategies for integration into a number of approaches, including co-simulation, code integration, model encapsulation, and model translation. Co-simulation lets partial models of a system be simulated in different tools. Once the model is validated through simulation, a final “production code” is generated through code generation, coordination (encapsulation), and integration into one system platform. The author of this paper extends the concept to propose a two-way integration strategy.⁸ However, these strategies have their own challenges, such as the synchronization of the simulation processes, automatic code generation, and so on. Generally speaking, not only are these efforts time-consuming, but also they are still facing technical difficulties to reach streamless integration (as we will give an example in Section III). Furthermore, the auto-generated code is oftentimes not sufficiently optimized to be used in a production environment. As the complexity of the system is increasing rapidly, these challenges become more and more evident.

In this paper, we present an interactive design and simulation platform for flight vehicle systems development. The proposed platform adopts the co-simulation concept and avoids code generation challenges faced by other integration approaches. It enables the component design “plug-and-play” in a systems simulation environment, therefore, to bring the “systems simulation” into the “control design” for integrated development. Moreover, the smooth interactive design and simulation is achieved by an adaptive “connect-and-play” capability. As a case study, this paper demonstrates an interactive development between an aircraft pitch tracking control design and flight simulation on a generic jet flight training device (FTD).

The rest of this paper is organized as follows. In Section II, we describe the interactive design and simulating platform in details. In Section III, we present the experimental test bed for the integrated flight vehicle systems development, followed by implementation (Section IV) and an interactive flight control design and simulation example (Section V). Finally, Section VI offers the concluding remarks.

II. Interactive Design and Simulation Platform

Modern aircraft include a variety of automatic control systems that aid the flight in navigation, flight management, and augmenting the stability characteristics of the airplane.⁹ The aim of flight control systems (FCS) development is to find a solution, given the inputs and desired outputs or tolerable errors, and to integrate design into a functional system that performs its assigned tasks satisfactorily. The FCS development process is typically broken down into several chronological stages that are extensively interrelated and interconnected,¹⁰ including the following critical ones: 1) Establishment of System Purpose and Overall System Requirements; 2) Detailed Component Design and Selection; and 3) Integration, Testing, and Validation. Due to its complexity, a much more integrated design process may be a natural choice of solution, where interactions among subsystems and components, interactions at different levels of systems complexity, and interactions across different phases can be accounted for, to reduce design iterations and become more robust and reliable. On the one hand, the design of different control channels are independent activities, with customized flight equations that capture the characteristic dynamic behavior associated with that specific channel, and with simplified interacting component models to minimize the coupling effects in control. On the other hand, the final designed controllers need to be implemented and evaluated under a realistic simulation environment, consisting of high-fidelity nonlinear aircraft systems models. The design is often tested by a series of systems simulations, including those performed on a flight simulator. In this paper, we propose an interactive design and simulation platform for flight vehicle systems development, especially for flight control systems design and verification.

The integrated development process is illustrated in Fig. 1. “Controller Design” refers to iterations of design and design modifications, including multi-paradigm models. The “systems simulation” part implies various levels of simulations as verification and validation methods, including heterogeneous platforms.

There are several integration strategies.⁷ The popular approach in flight control systems development follows the concept of *code encapsulation* in principle. As shown in Fig. 2(a), the controller is designed and validated in isolation by desktop off-line simulation. The controller algorithm codes are generated (in C code as one example). Then, encapsulated as a monolithic submodel, it is integrated into the model of enclosing system for systems validation and verification. Such process presents the part from “Controller Design” to “Systems Simulation” of the integrated process in Fig. 1. In our paper,⁸ we extended the approach by further suggesting a two-way integration strategy, as shown in Fig. 2(b). This approach differs from the one-way code encapsulation approach (Fig. 2(a)) in that it includes

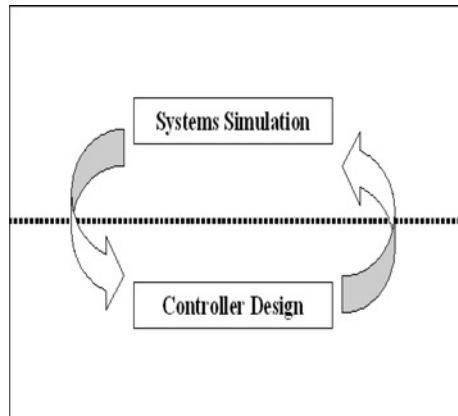
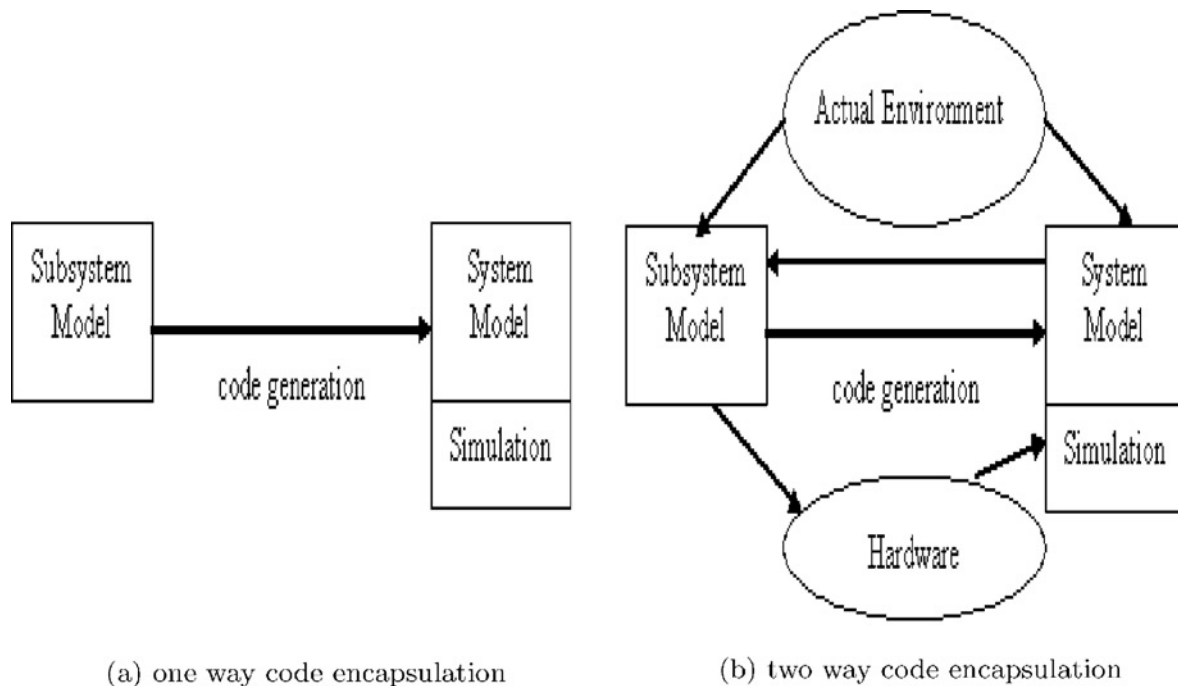


Fig. 1 Integrated (FCS) development process.

the hardware-in-the-loop, heterogeneous simulations, and it enables “Systems Simulation” → “Controller Design” capability in the integrated process (Fig. 1).

Still, the challenges of the code-generation based integration approaches include the level of automation, compatibility and synchronization of the models and simulations. Since design codes are used as the media, one needs to make sure that the generated codes can work on heterogeneous simulation platforms, with proper interfaces. If so, it is expected that the code generation can be processed automatically, to avoid tedious manual labor and errors. These challenges are still open research topics. In this paper, instead, we adopt a different integration strategy that is similar to *co-simulation* in principle. As shown in Fig. 3, this proposed interactive platform allows the component model to be simulated (plug-and-play) in a different, system-level environment. Moreover, the platform is adaptable such that the systems simulator can “connect” to the design model directly. We believe that this “connect-and-play” capability is one significant improvement over the “plug-and-play” capability. Since there is only one physical design



(a) one way code encapsulation

(b) two way code encapsulation

Fig. 2 Code-Generation based integration approaches.

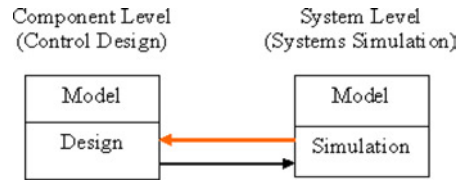


Fig. 3 Interactive design and simulation platform.

model that takes residence at the component level, one can work with this model to make modifications and perform testing “on-line” without the intermediate code-generation process. Obviously, the “connect-and-play” property and adaptability make the design and simulation platform truly interactive and integrated in development.

Under the proposed platform (Fig. 4), the integrated flight control system results in a multi-paradigm control framework. It represents a standard flight control system block diagram with some special features.

- The blocks with a drop-down shadow represent “swapping” features. The guidance and command block represents the flight path generation (guidance) or command inputs (for controller design). The actuation and sensor blocks can be replaced by software modules with different levels of fidelity, or even hardware equipments. The vehicle dynamics module can also be replaced by different software modules for different simulation purposes. A simplified linearized dynamics model is used for control system design, while full-scale nonlinear flight equations will be used for high-fidelity simulations, such as flight simulations.
- The blocks inside HIL can be replaced by hardware equipments for hardware-in-the-loop (HIL) experimentation.
- The whole flight control system structure, when interacting with other flight systems, can be integrated into a flight simulator for flight simulation, or pilot-in-the-loop (PIL) simulation, to validate the design.
- In order to emulate the reality that different flight systems components are physically installed in different locations and their interactions are communicated through mechanical links or electrical bus, the proposed framework allows for a distributed modeling structure. Each block can be individually modeled, as one software module in different processors. Therefore, it is possible to distribute different parts of a computing task across individual processors operating at the same time, or “in parallel,” and thus reduce the overall time to complete the task. Further, the distributed modeling structure makes it feasible to “swap” different modules of the same block, including the hardware-in-the-loop simulation.
- Due to the distributed modeling and “swapping” feature, it is possible to replace block modules developed under different platforms, and even to run simulations on machines from different manufacturers. Therefore, the proposed framework supports heterogeneous simulations.

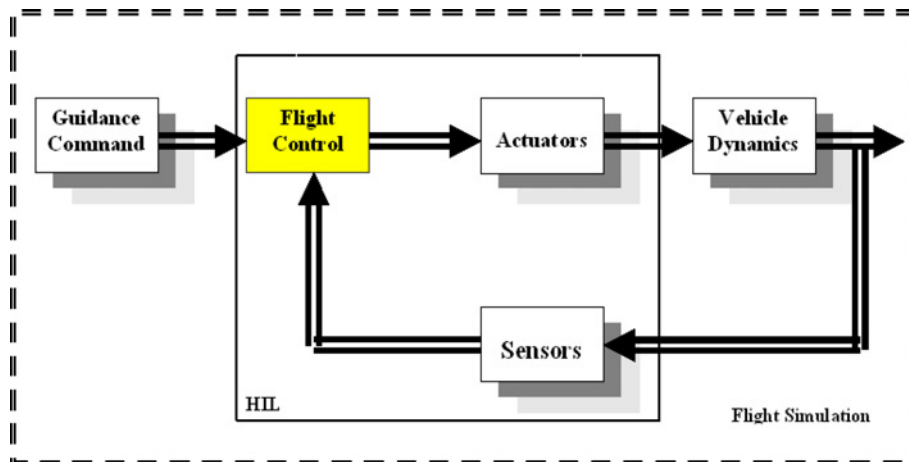


Fig. 4 Multiparadigm FCS block diagram.

- The framework not only allows for distributed modeling, but also enables real-time simulation, where interactions and synchronizations among subsystems or components act and react in clock time, as it happens in a real flight environment.

In order to demonstrate the proposed FCS framework and the interactive design and simulation platform, an experimental test bed is set up. A real-time systems simulator and a flight training device (RTSS-FTD) are equipped to provide a suitable proof-of-concept facility, and illustrated by a pitch tracking control example of a generic jet airplane in its cruise condition.

III. RTSS-FTD Test Bed

A. The Real-Time Systems Simulator (RTSS)

The real-time systems simulator (RTSS) facility is a networked cluster of high-end commercial off the shelf (COTS) computers as shown in Fig. 5(a). Its core computing features include: three (3) host computers each has dual-Pentium-processors running Windows 2000 OS; four (4) real-time computers each has dual-Pentium-processors running QNX real-time operating system; the real-time nodes are directly connected by 400 Mbit/sec FireWire and communicate with hosts over a dedicated 100 Mbit/s Ethernet network. Further, the system consists of 108 multiple channels IO system for hardware-in-the-loop simulation. The RTSS is also connected through a 1.25 Gb/sec Giganet to a similar facility to share data and sources, and it is connected to a 56-alpha-processor high power computer for off-line computing and simulation, as well as data storage. This configuration provides the following key capabilities to support our proposed framework:

- **Flexibility.** The models are distributed and executed over a network of high-end computers interconnected with a fast real-time communication system; the data is real-time acquired, logged, and stored; the model parameter values can be modified at runtime remotely from a graphical interface on the host computers; the interconnection with the commercial I/O board is located inside one host computer, allowing for hardware-in-the-loop simulation.
- **Development.** The development environment uses the commercial Matlab/Simulink platform[‡] with an add-on package called RT_Lab[§] for real-time hardware-in-the-loop simulations. It provides a full integration with visual simulation, as well as the capability of C code generation. To meet for higher computational demands, parallelization can be performed by splitting the model in several interconnected subsystems. The automatic code generation and object code loading take into account of all necessary processors and I/O synchronizations. A library of Simulink icons can easily connect commercial I/O boards to the dynamic models. An application programming interface (API) provides a user-friendly interface to allow for the control of the simulator, the on-line parameter control and results display. It also generates the source codes and typical I/O drivers and real-time modules allowing the user's addition, specialization, and customization. In conclusion, the RTSS itself provides a stand-alone modeling and simulation development capability that enables both one-way and two-way interactive platforms as we discussed previously. The application of integration and code generation is also presented in the authors' earlier work.⁸ However, as mentioned before, the automatic code generation is still a challenging task. For example, third party network socket libraries are needed on Windows and QNX during our project implementation. Moreover, the automatically generated C codes require fine tuning before its implementation, or at least are not optimized yet. In this paper, we propose the "connect-and-play" concept to totally avoid the code integration. In other words, the code generalization feature of the RTSS is NOT utilized in this presentation. In cases where the simulation accuracy or model complexity demand for faster model execution, the C code generation can still be used.
- **Scalability.** The system takes advantage of COTS hardware and components to fit the application requirements, and to expand the computing power if necessary.
- **Performance.** Fire Wire 400-Mbits/second real-time serial bus offers a very low latency for models with loop time as low as 200 ms. The scheduler overhead is less than 10 ms on a Pentium 233 MHz processor. The minimum loop time on a distributed CPU system is about 80 ms to account for data synchronization and

[‡] www.mathworks.com

[§] www.opal-rt.com



Fig. 5 RTSS-FTD facilities.

TCP/IP communication with the host computer. The use of QNX proves a 2 ms interrupt respond time and a 6 ms context switching time.

B. The Generic Jet Flight Training Device (G-FTD)

A separate flight training device (FTD) is also set up for flight simulation, as shown in Fig. 5(b). This state-of-the-art research simulator simulates the operation of a generic jet aircraft within the tolerances and conditions set out by the Transport Canada Authority. The block diagram of Fig. 6 is a general overview of the system layout. The major aircraft subsystems include the automatic controls, the auxiliary power unit (APU), doors, the engine indication

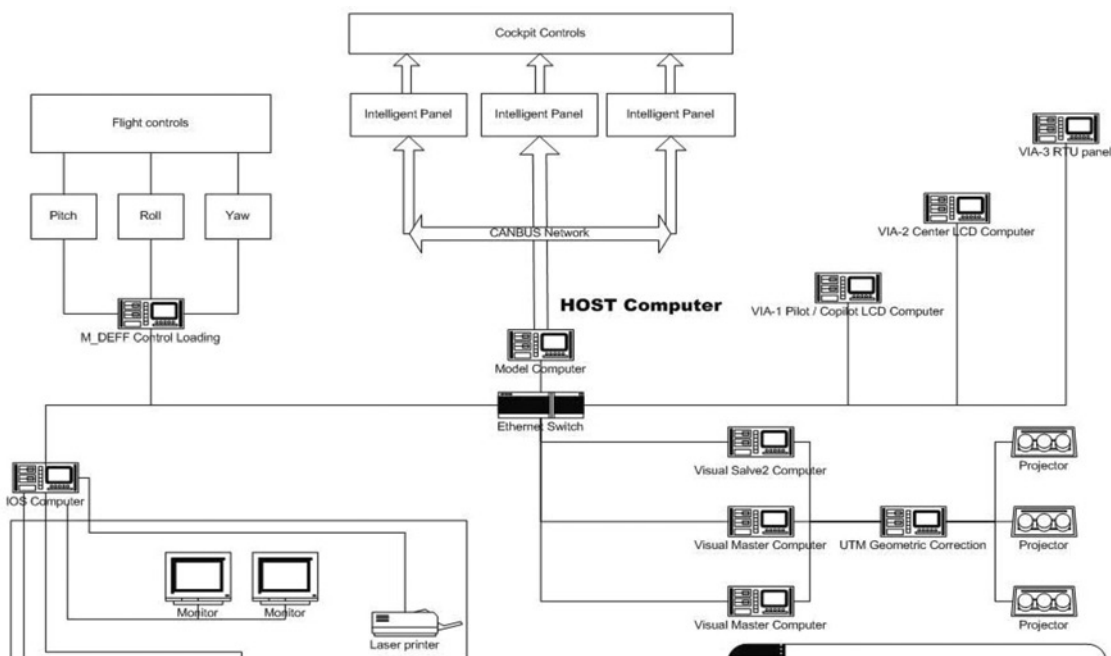


Fig. 6 G-FTD architecture.

and crew alerting system (EICAS), the electrical systems, the environmental control systems, flight controls and flight instruments, the fuel, pneumatic and hydraulic systems, the landing gear, the lighting, and the navigation and communications systems. The design of the FTD is such that all the simulated functionality is concentrated in the software model running on the host computer. This software model contains all the mathematical and logic modeling to make the FTD behave like the Generic Jet aircraft. All the other computers and hardware are input/output (I/O) interfaces between the pilot/copilot and the model software running on the host computer. The control loading is handled by a PC on the network. It communicates with the host on the Ethernet switch. This computer has digital wiring running to the primary flight controls in the cockpit. The computer systems are networked through a 100 Mbaud Ethernet switch. All the simulated aircraft panels are intelligent; they each contain an embedded CPU which manages their local IO and communicates with the host computer through a CAN bus network. The aircraft flight and subsystem models are developed using C language. The visual database is developed using the MultiGen paradigm. The control system is developed under the Matlab/Simulink platform.

C. The RTSS-FTD Connection

In summary, the RTSS is able to simulate the aircraft systems and flight maneuvers. The features of reconfigurability, modeling and customization of cockpit displays are critical to our systems integration research. Matlab/Simulink is the software development platform. The G-FTD presents a more complete and realistic aircraft model, which includes factors not taken into account in the RTSS development. It offers a different perspective as the flight mission may be observed from a cockpit with out-the-window visual and instrument displays. The RTSS and FTD facilities are connected through Ethernet cables to form a networked RTSS-FTD test bed for integrated modeling and simulation activities.

IV. Implementation of Interactive Platform

The FTD consist of a number of computers with highly specialized tasks. The core of the system is formed by the *Model* computer, which is responsible for the overall aircraft model, and the *Control Loading* computer. Three *Visual Instruments Aid* (VIA) computers are in control of the avionics instrument panels. Three *Visual* computers are in charge of evaluating the visual databases, each one creating an image of the current view outside the cockpit. Those images undergo blending and geometric correction algorithms in the *Universal Transverse Mercator* (UTM) correction computer and are finally projected onto a single curved screen. The simulator can be controlled from the *Instructor Operation Station* (IOS) computer. The *Model* and *Control Loading* computers, subjected to hard real-time constraints, run the real time operating system QNX, while all other computers have Windows 2000 installed. This system architecture forms a separated local area network. The *IOS* computer, however, has two network cards installed, allowing for the possibility to connect the FTD to another network. Figure 7 gives an overview of this layout.

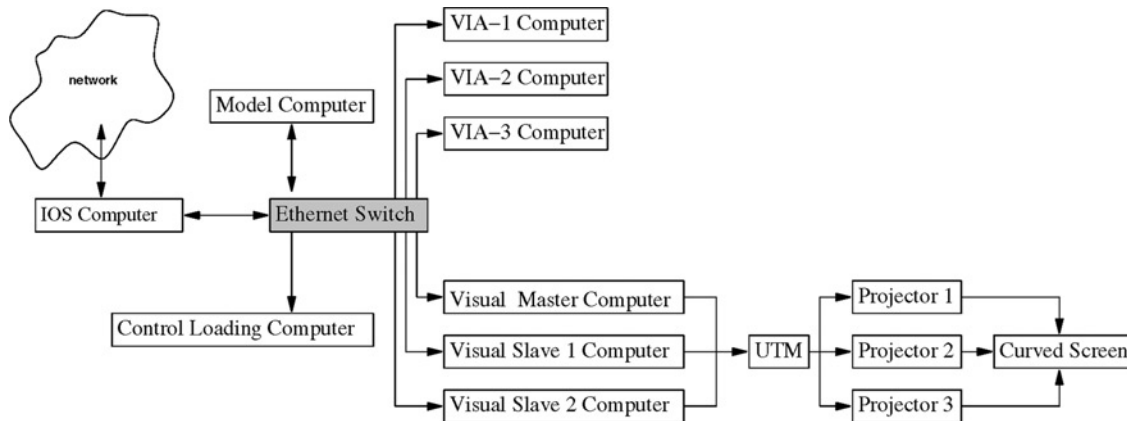


Fig. 7 FTD network layout.

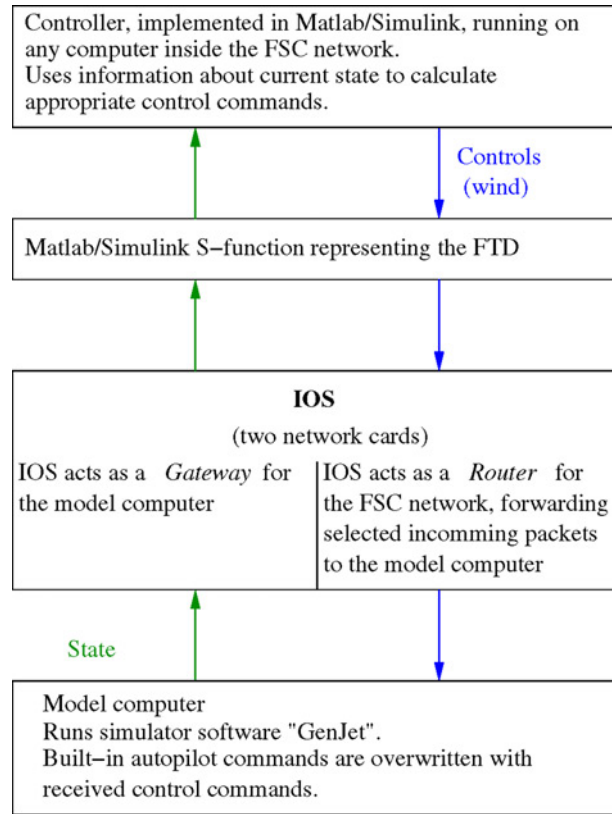


Fig. 8 Network communication.

In order to use the FTD as a test bed for interactive controller design and simulation, a network connection is established for “connect-and-play.” As introduced before, the control development environment uses the Matlab/Simulink platform which offers an application programming interface called “s-function” that can be used to integrate blocks with user-defined behavior to a Simulink block diagram. The idea is to let an S-function for Matlab/Simulink work as a network I/O-layer, which outputs the current state vector of the FTD and takes control commands as inputs, as it is commonly done with hardware-in-the-loop approaches (Figs. 8 and 9). Note that the control input can carry additional payload, if necessary. In particular, the S-function allows for three values of wind components, which, if given, will be used to overwrite the built-in wind model.

The communication link between a Matlab/Simulink environment and the FTD hard-/software has to fulfill a number of requirements. Mainly, the interface should be

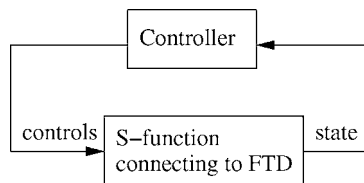


Fig. 9 Integration in Matlab/Simulink.

- easy to use no compilation should be required when switching between the built-in and a remotely controlled autopilot
- robust the communication should not break down on minor network problems
- fast the packets should have low overhead to improve network speed
- capable of real time maximum response times should be guaranteed
- extensible it should be easy to integrate additional payload
- portable the packet format should not depend on operating system peculiarities

It is then decided that a TCP/IP connection offers the robustness needed for controller operation. The packet format used for the network connection is simple, yet extensible. A fixed-length integer specifying the total packet length is followed by an arbitrary number of triples specifying an identifier, a value and a delimiter. The S-function is written

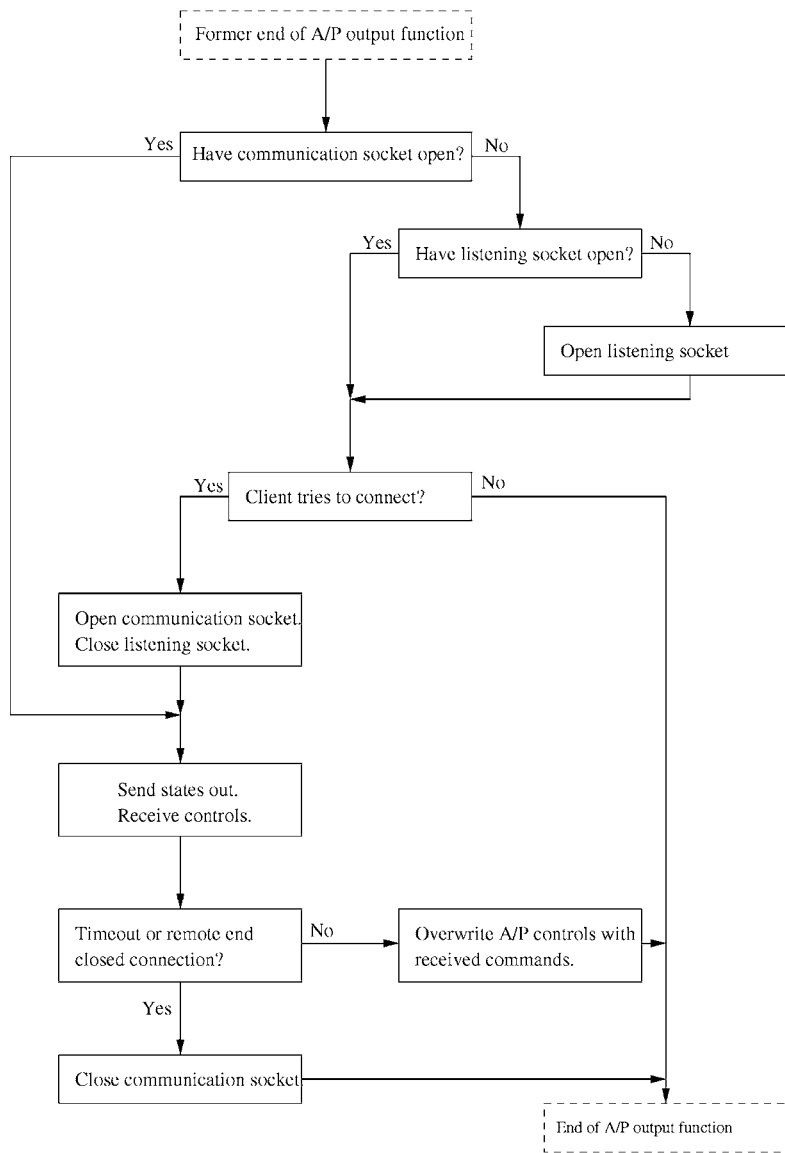


Fig. 10 Connection logic.

in the programming language C, and can be further customized at compile time using preprocessor macros. This way, among other preferences, maximum allowable packet length and floating point data formats can be adjusted.

In order to guarantee maximum response times of the network connection, network timeouts have been implemented, also settable with C preprocessor macros. If a timeout occurred, predefined hard coded default values are assumed. In this case, the user is provided with a warning.

One of the demands for the connection is that it should be possible to switch between the built-in and a remote controlled autopilot “on the fly.” This required the logic depicted in Fig. 10, which resides on the FTD side of the network connection. In practice, a custom autopilot takes over as soon as a Simulink simulation containing the network S-function is started, while in general, the built-in autopilot is in place while the simulation is not running.

V. A Pitch Tracking Example

In this section, a pitch tracker for a jet airplane at its cruise condition is considered, with elevator deflections being used as the control input. In early design stages for aircraft controllers, the influence of actuator dynamics is oftentimes neglected. In this case, however, this turns out to be an unjustifiable simplification. Figure 11 shows the simulation results of a LQR (linear quadratic regulation) based pitch tracker, acting on a linear aircraft model with unmodeled actuator dynamics. LQR is a design method to achieve an optimal objective function, often representing the “energy” of the system. On the left side, the pitch angle (solid line) follows the trajectory (dashdot line) nicely, while the control effort, depicted on the right, is rather small. When this controller is tested with the FTD in the loop, it becomes unstable; the simulation shown in Fig. 12 has to be aborted after 30 seconds. On the right side, the commanded elevator position δe_c is plotted as a dashdot line against the current elevator position δe . This leads to the assumption that the instability is caused by the time-delay introduced by actuator motion. Therefore, the

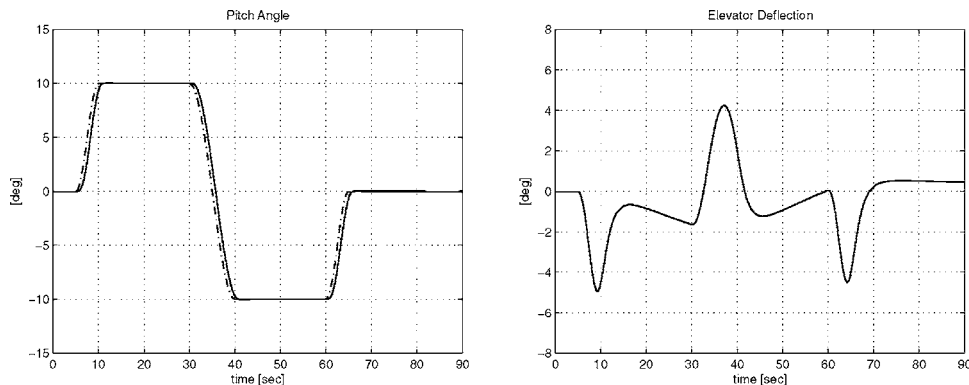


Fig. 11 Off-line Simulink simulation, no actuator dynamics.

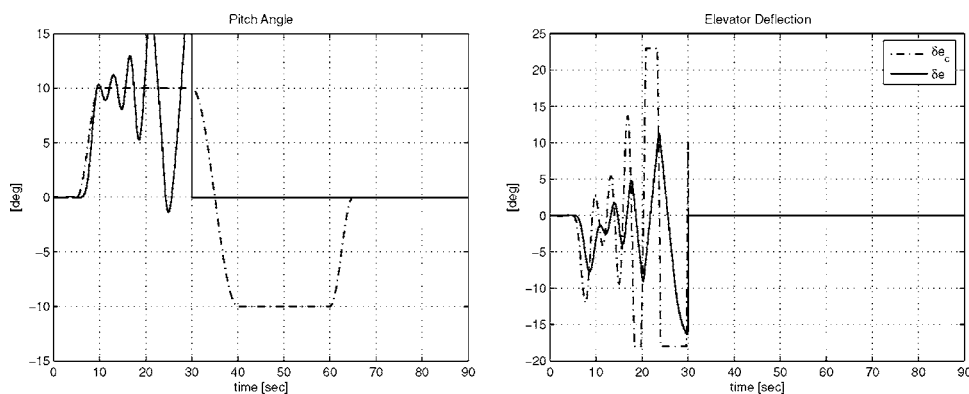


Fig. 12 FTD in the loop simulation, no actuator dynamics.

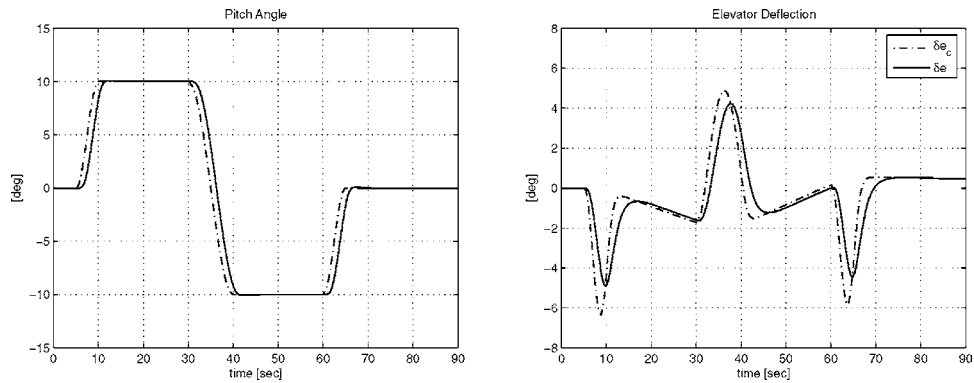


Fig. 13 Off-line Simulink simulation, re-designed controller.

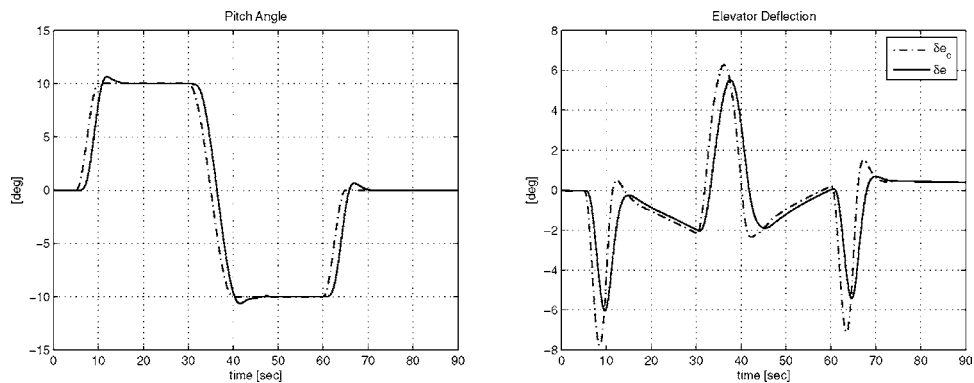


Fig. 14 FTD in the loop simulation, re-designed controller.

actuator was modeled as a first-order lag filter, and included in the linear aircraft model during controller re-design. Simulation results with the linear model and the FTD in the loop are shown in Figs. 13 and 14. Although Fig. 14 still shows a slight overshoot, the improvement over Fig. 12 is obvious.

In this example, we demonstrate that the flight control design and its verification achieve the expectation by using our proposed interactive (control) design and (flight) simulation platform. Design modifications or even redesigns are quick to perform since the simulation directly “call” the component that still takes residence at the local control system development environment (Matlab/Simulink). The design impact on overall system performance is also easy to obtain by the interactive platform due to its “connect-and-play” capability. One doesn’t need to go through the code generation and interfacing process.

VI. Conclusions

In this paper, an interactive design and simulation platform was presented. Its “connect-and-play” capability and adaptability enabled “on-line” interaction between design and simulation during the integrated development. The implementation of the proposed platform was addressed in details, and an aircraft flight control system development example was given as a demonstration on an experimental RTSS-FTD test bed. Multi-paradigm models and heterogeneous simulations were integrated through a TCP/IP network of I/O. This platform is deemed to attract interests in development due to its simple strategy. The interaction and integration can be achieved without “re-inventing” machine or architecture.

As the complexity of system increases rapidly, the modeling and simulation complexity will inevitably grow with the development stages or phases as they reflect ever-increasing accuracy in requirements. The proposed “connect-and-play” strategy presented in this paper was demonstrated in an example with simulations integrating early-stage controller development and final-stage “flight testing.” In this example we dealt with the situation when both development platform and flight simulation platform are available. Nevertheless, the concept is general and it can be applied to other engineering development environment with models of different level of accuracy and simulations of different level of fidelity. If the developers have an opportunity to build a simulator, it will help to keep such a “connect-and-play” feature in mind during their design. With proper communication protocol and interface selections, the modeling and simulation activities will become smoother in their transitions and interactions.

Acknowledgment

The authors would like to express their appreciation to the editors and the anonymous reviewers for the constructive comments and suggestions.

References

- ¹Heck, B. S., Wills, L. M., and Vachtsevanos, G. J., “Software Enabled Control: Background and Motivation,” *Proceedings of the American Control Conference*, Arlington, VA, 25–27 June 2001, pp. 3433–3438.
- ²Wills, L., Sander, S., Kannan, S., Kahn, A., Prasad, J., and Schrage, D., “An Open Control Platform for Reconfigurable, Distributed, Hierarchical Control Systems,” *Proceedings of the Digital Avionics Systems Conference*, Philadelphia, PA, October 2000.
- ³Paunicka, J. L., Mendel, B. R., and Corman, D. E., “The OCP—An Open Middleware Solution for Embedded Systems,” *Proceedings of the American Control Conference*, Arlington, VA, June 25–27 2001, pp. 3445–3450.
- ⁴Mosterman, P. J. and Vangheluwe, H., “Guest Editorial: Special Issue on Computer Automated Multi-Paradigm Modeling,” *ACM Transactions on Modeling and Computer Simulation*, Vol. 12, No. 4, October 2002, pp. 249–255.
- ⁵Mosterman, P. J., Sztipanotits, J., and Engell, S., “Computer Automated Multiparadigm Modeling in Control Systems Technology,” *IEEE Transactions on Control Systems Technology*, Vol. 12, No. 2, March 2004, pp. 223–234.
- ⁶Liu, J., Eker, J., Janneck, J. W., Liu, X., and Lee, E., “Actor-oriented Control System Design: A Responsible Framework Perspective,” *IEEE Transactions on Control Systems Technology*, Vol. 12, No. 2, March 2004, pp. 250–262.
- ⁷Müller-Glasser, K. D. and Frick, G., “Multiparadigm Modeling in Embedded Systems Design,” *IEEE Transactions on Control Systems Technology*, Vol. 12, No. 2, March 2004, pp. 279–292.
- ⁸Liu, H. H., “Multiparadigm Design, Validation and Verification by Simulation in Flight Control System Development,” *Proceedings of IEEE Conference on Computer Aided Control Systems Design*, Taipei, Taiwan, ROC, 2–4 September 2004, pp. 71–76.
- ⁹Nelson, R. C., *Flight Stability and Automatic Control*, WCB McGraw-Hill, 2nd ed., 1998.
- ¹⁰McRuer, D., Ashkenas, I., and Graham, D., *Aircraft Dynamics and Automatic Control*, Princeton University Press, 1973.

Stanley Nissen
Associate Editor