DUECA - Data-driven activation in distributed real-time computation

van Paassen, M.M.; Stroosma, O.; Delatour, J

DOI
10.2514/6.2000-4503

Publication date
2000

Document Version
Final published version

Published in
AIAA Modeling and Simulation Technologies Conference

AIAA Modeling and Simulation Technologies Conference and Exhibit
14-17 August 2000 Denver, CO

A00-37286

AIAA-2000–4503

DUECA - DATA-DRIVEN ACTIVATION IN DISTRIBUTED REAL-TIME COMPUTATION

M.M. (René) van Paassen* and Olaf Stroosma†
Delft University of Technology, PO box 5058, 2600 GB Delft, The Netherlands
J. Delatour‡
LAAS-CNRS 7, avenue du Colonel Roche 31077, Toulouse Cedex 4

Experiments and flight simulation programs for a research flight simulator can be considered as a combination of real-time distributed calculation processes. Current tools and architectures for implementing these processes are complicated to use and require considerable real-time programming skills. The created programs are not sufficiently flexible. A new middleware layer, DUECA, was developed to facilitate implementation of programs on a research flight simulator. It combines a publish and subscribe mechanism and message passing facilities with some novel elements, namely explicit allocation of process activation and the conditions under which that allocation should take place, synchronisation of data from different sources, and a mechanism to transparently combine processes running at different update rates.

Introduction

THE Delft University of Technology makes use of flight simulation for research, and the research of flight simulation techniques. The foundation of the international institute for research into SImulation, MOtion, and NAvigation, and the construction of the SIMONA Research flight Simulator (SRS) are prime examples of this activity.

Version 2 of the SRS will become operational in May of 2001. This version will contain an elaborate display system and an advanced control loading system. The SIMONA simulator consists of a light-weight carbon and aramid-fiber aircraft cabin mounted upon a high-performance six degree of freedom motion platform (Figure 1).

The low mass and the stiff construction of the cabin, together with the high motion bandwidth of 15 Hz enable high quality simulations in which the pilot will not be aware of any delays between his input signals and the response of the cabin. Such a high frequency response requires a high update rate

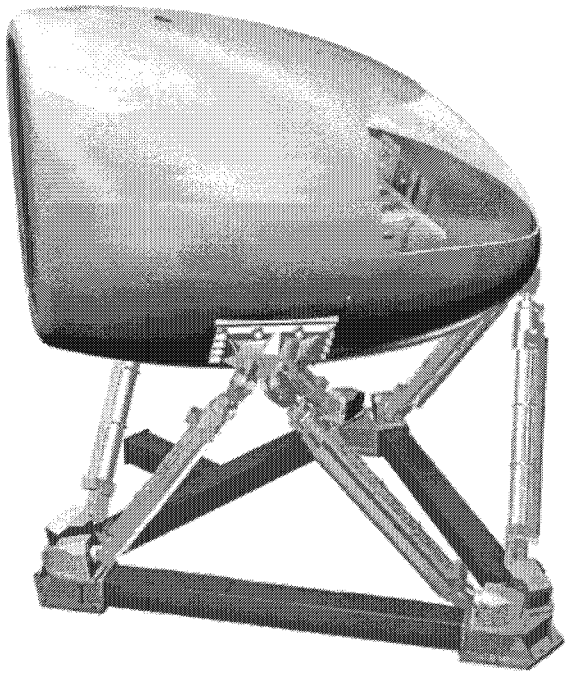

Fig. 1 SIMONA Research Simulator

*Assistant Professor, Faculty of Aerospace Engineering
†Research assistant, SIMONA Research Institute
‡Doctorate Student, Laboratoire d'Analyse et d'Architecture des Systèmes

and near zero delay of the signals controlling the platform. This update rate can be produced by a real-time computer system especially developed for SIMONA.

Additional information on SIMONA can be found at the following location: `http://www.simona.tudelft.nl`.

The computational infrastructure for the SRS consists of several computers, linked by a high speed communication network (SCRAMNet [1]). This network provides a fiber-optic link between the computers, a global memory space accessible to all computers and interrupt facilities.

A simulation or experiment can be seen as a real-time distributed calculation process, and the following functions can be identified in this process:

- Synchronization of calculation processes with wall clock time
- Communication of data between processes; this communication may cross nodes in the real-time network.
- Matching of calculation processes with communication; i.e. the prevention of race conditions.
- Communication with the simulator hardware.

An experiment with the SRS hardware will require running synchronized real-time programs in several of the computers, and, without advanced programming aids, development of such an experiment is a task that will require considerable expertise on the field of real-time programming and computer communication.

Design and implementation of real-time programs on distributed hardware is normally the work of experts. Especially the design of a communication and activation schedule, and testing the performance of a real-time process is a complicated job. However, the users of the SRS will be scientists and students interested in solving a problem in simulation or human-machine interaction, and are usually not experts in real-time distributed programming. A software environment was therefore needed that supports these users in the development of real-time experiments and simulation. The planned environment consists of two parts:

- A middleware layer, DUECA (Delft University Environment for Communication and Activation). This layer facilitates the development of real-time distributed processes, by hiding the distributed nature of the network, and by providing synchronization of calculating processes.
- An application framework, DUSIME (Delft University SIMulation Environment). DUSIME provides basic capabilities needed for a real-time simulation, such as run control facilities (start/stop, etc.), initial condition calculation support, snapshot taking and logging facilities.

Since experiment development and modification will be a more or less continuing process in SIMONA – as with any research simulation facility – the simulation programs and experiments will be subject to frequent change, modification and extension. DUECA and DUSIME are designed to support rapid prototyping and extension and/or modification of simulation programs.

Existing approaches

Some existing approaches for the implementation of real-time simulations on distributed computing hardware were reviewed:

- The "traditional" use of a common data area (sometimes called data dictionary) that is globally accessible, and a schedule or calling sequence for the calculations, e.g.[1] A drawback of this approach is that schedules for distributed, multi-node calculations are difficult to implement, and very inflexible. A careful design of the schedule, based on data use and production of all calculation processes, must be combined with hand-programmed semaphores (or other synchronization mechanisms) to guard against race conditions. Errors in the design of the schedule are easy to make, leading to time-related imperfections in the implementation of the model. The advantages of this approach is that it has little run-time overhead.
- The use of message passing mechanisms. In this approach the scheduling of processes in the system is triggered by the arrival of data. This eliminates the necessity of a (hand crafted) schedule, and it solves the problem of race conditions. In a traditional message passing application (e.g. with the tools supplied by QNX[3]), the destination of messages has to be known to the sender. Disadvantages of this approach are that the run-time overhead is larger, and the implementation is still not very flexible, e.g. additional processes require modification of the code that sends out data. A more important disadvantage is the programming overhead of the model. Calculation processes for simulation experiments usually require data from multiple sources. In this solution the application

[1] Reg. Trademark of Systran Corp.

programmer has to provide the code that *synchronizes* data with the data coming in from other sources. The advantage is that the schedule is generated by the data communication.

- Message passing in combination with a publish-subscribe mechanism.[6] Examples are the use of real-time Corba,[5] or HLA for distributed simulation.[2] This makes the message-passing solution more flexible. A disadvantage is that the programming overhead is increased.

None of these approaches satisfied our needs to the desired extent. Therefore it was decided to develop a new middleware layer, DUECA, short for Delft University Environment for Communication and Activation.

DUECA

DUECA is specifically designed for the implementation of simulations and simulator experiments. It uses a publish-subscribe mechanism to set up the communication, by which it supports a modular and flexible program design. In addition, the following functions have been added, with the aim of facilitating the implementation of a distributed real-time computational processes:

- Activation is allocated explicitly in DUECA. In a normal message passing system, activation occurs for each incoming message individually. In DUECA activation can be coupled to a combination of messages, so one can specify that a module in the simulation is only activated when *all* inputs needed by that module are available.
- DUECA can wait for a synchronous set of data, and only then activate a process. Even when incoming messages run "out of synchronization", e.g. data from different times is simultaneously available, DUECA can provide a time-consistent set of messages to a user's module.
- Mechanisms are provided to transparently work with data that has different update rates.

This provides a flexible environment for the implementation of simulations and experiments. The simulation can be programmed as a set of modules connected via data channels, in a data-flow architecture. A module in DUECA will be written as a (C++) class, and the steps typically needed at the creation of a module are:

- Publication and subscription is done to obtain a connection to the output and input data channels.

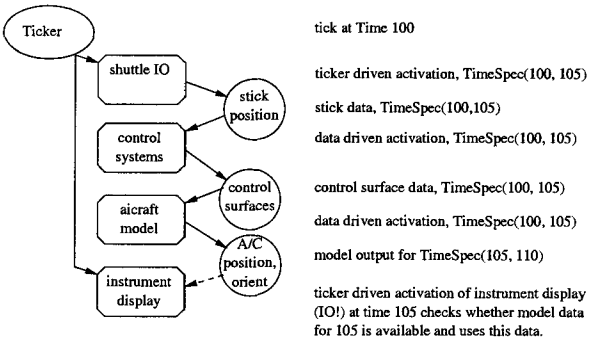

Fig. 2 Illustration of the data-driven activation in DUECA. Three processes are time driven (by the Ticker), input, output and atmosphere calculations. Other processes are activated by the (possibly joint) availability of data.

- An activity object is created.
- This object is linked to a method of the class that implements the activity, i.e. the model calculation.
- The update rate of the activity may be specified.
- The triggering condition for the activity is specified as the simultaneous availability of data on the input channels.

The method that implements the activity is then invoked each time a model update has to be calculated. Upon invocation, this method can access a time-consistent set of data from the input channels, perform the model calculation and send the result over the output channels. By thus hiding the synchronization issues associated with a real-time program, application programmers can use data flow diagram concepts, familiar from control systems engineering and modeling environments such as Matlab/Simulink.

The use of a scripting language (Scheme,[4]) and a simple interface between the scripting language and the modules provides added flexibility and re-use; a simulation or experiment can be composed from existing modules and tailored in a creation script written in Scheme.

Basic concepts

The programming model introduced by DUECA uses the concept of "module" for the components supplied by application developer. A module is a software entity that can use the services supplied by DUECA to communicate with other modules. A number of service levels can be distinguished in DUECA:

DUECA base The base service level in DUECA is accessible to modules written in (currently) C++. This level provides;

1. Naming, identity and registry. Each module using DUECA services must have a unique name. By deriving from the parent NamedObject (usually not directly, but indirectly), this service is provided. In addition to the name, DUECA assigns an Id, composed of a locationId, pointing to the DUECA executable the module lives in and an objectId, which is unique within the DUECA node.

   A pointer to the module (of type NamedObject*) can be obtained by querying DUECA with a name or with a module Id.

2. Communication. DUECA modules can communicate by means of "Channels". A channel is a (possibly) distributed communication device for communicating one specific type of data. Channel versions exist for communicating events and for communicating stream data; i.e. data that is refreshed regularly.

3. Activation. A computational process in DUECA does not require the specification of a global schedule (in the form of: first module A, then module B, then modules C and D, etc.) instead each module describes the conditions under which it has to be scheduled. This can be simple, for example many modules implement a calculation like:

$$x(t+1) = F(x(t), u(t)) \quad (1)$$

$$\text{or:} \quad y(t) = G(x(t), u(t-1)) \quad (2)$$

   For module F, the activation condition is that the data x and the data u are both available at a certain time t. It can then produce data x for time $t+1$. For module G, data x has to be available at time t and data u at time $t-1$, it can then produce data y for time t. For real-time simulation, the activation can also be requested from a "Ticker", which provides activation at regular intervals. See the example in Figure 2.

DUECA configuration DUECA has the capability to implement a computational process in a distributed manner, on a number of computers. The channels in that case will have multiple channel ends, one end in each executable in which that channel's data is read or written. To use DUECA on a distributed platform, one must start a DUECA executable on all the platforms. Each executable must have a unique LocationId, and one must specify how the data is to be transported to the other DUECA parts.

The programming language "Scheme" is used to implement the DUECA configuration. Special scheme commands were added to the interpreter used by DUECA. These commands can be used to set up the transportation media used and to specify which partners get their data by what media. A file "dueca.cnf", in the directory in which the DUECA executable is started, is read and used to obtain configuration data.

DUECA creation DUECA provides an elaborate and flexible system for creating so-called "entities" from modules. An entity is for example an aircraft in a simulation. A DUECA system can contain and update several entities in parallel.

The programming language Scheme is also used to implement the creational process. A scheme script can for example describe the composition of an entity out of several modules. If the modules require or can accept parameters, it is possible to specify these parameters in the script. An example of simple entity is:

```
(define ent
  (make-entity
   "PH-C0Z"
   (list-of-modules 0
    (make-module 'c172-dynamic-model
'initial-position "57.34.18N05.23.06E"
'initial-heading 330)
    (make-module 'c172-ai-pilot))))
```

This would make a Cessna 172[2]. The complete model is programmed in a single module, and it is combined with a module for an artificial intelligence pilot. The plane's initial position is given in a string with earth coordinates, and the initial heading is 330. If the artificial intelligence pilot has any sense she will start taxiing, flying etc., when the simulation is started.

DUECA control This service level can be one of many versions. It provides a human user (op-

[2]Since Scheme is a full-fledged programming language, one could also make procedures that take a name and initial position, and create a Cessna. Add a loop and you can create a whole platform full of Cessna's

erator) the control over the entities in the computational process. The first implementation of this level will probably be an implementation for control of simulations, called DUSIME (Delft University SIMulation Environment). It provides the means to start, stop, take snapshots, replay, etc. Implementation of DUSIME is currently underway.

Activity

In order to play its role in a simulation, a module has two basic needs:

communication In order to perform the model calculations or output to the simulator hardware, a module has to gain access to data produced by other modules, and output data has to be delivered.

activation Especially in a real-time program, the timely execution of the model calculations is important. In general, a module needs to be "called" at a fixed update rate.

In a conventional real-time simulation program, the communication is ensured because parts of the program have access to a common data storage in computer memory, and activation takes place when "entry points", in the form of subroutine calls. In publish-subscribe systems, the communication is provided by a middleware layer. Usually a module requesting data can provide a function (or object method) that gets called when new data is available. The activation is thus triggered by the availability of data.

Allocation of Activity

In the decomposition of a simulation or experiment program, one can identify a number of activities, in which each of these activities consists of a logically coherent set of programming instructions. Some examples of activities are; - the calculation of aerodynamic force and moment contributions of an aircraft model, reading of the primary control inputs, generation of an EFIS display image.

In a conventional simulation program, a number of activities is strung together into a task. In this manner, the start of an activity is triggered by the completion of a previous activity. The problem is that for most activities, completion of a previous activity is *not* an adequate trigger. For most activities, and for all activities involved in updating the model state, the only adequate trigger is the availability of all the necessary input data. In a single-processor, single-thread simulation, one can determine a call

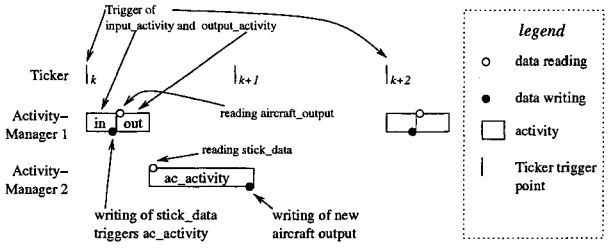

Fig. 3 Sequence of activities for a simple simulation, with time-driven input and output of data, and a data-driven model update.

order in which the completion of the previous activity *coincides* with the availability of the necessary data. For a distributed computation environment, this task quickly becomes complex, while synchronization mechanisms and data buffers need to be introduced.

A module in DUECA may explicitly define one or more activities. An activity can be triggered by various events. One possibility is triggering of an activity on the passing of wall clock time. This is appropriate for activities that implement input or output to the simulator's devices. An activity can also be triggered by the availability of data. If a module uses two or more data sources, it can specify that its activity should be triggered by the simultaneous availability of data from several sources. So the following steps are followed for connecting a module in DUECA:

- Subscribe to all data channels needed for the module.
- Create an Activity, and link it to a method or function that implements the Activities calculations.
- Specify that the Activity should be triggered when the necessary data is available.

The time associated with the data that is being published is stored in the channels. This time stamp is used in the synchronization of data from different sources. As an activity is (re-) started, it is invoked with a time specification as one of its parameters. Using this time specification, the activity can obtain a time-consistent set of data from the channels it reads its data from, and the data produced by the activity can be time-stamped with the same time specification. A simple example of a schedule created by specification of activities triggered by time and activities triggered by data availability is given in Figure 3.

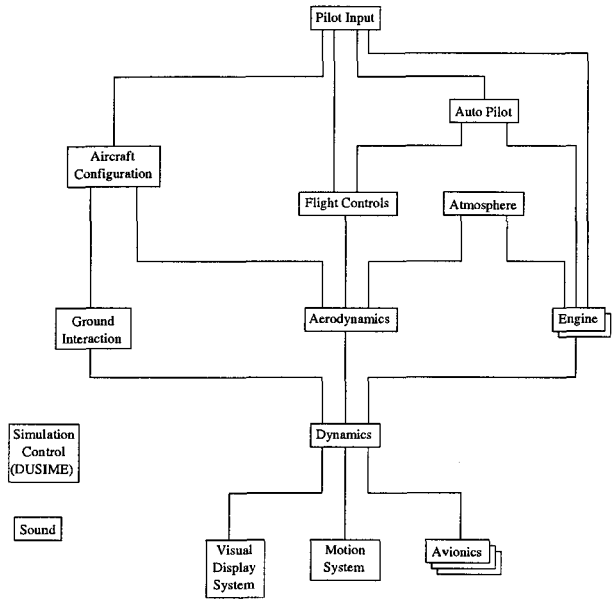

Fig. 4 Basic simulation architecture for an aircraft model in the SIMONA Research Simulator

Example implementation

This section discusses an example implementation of a – fairly basic – aircraft simulation in DUECA. It covers the basic architecture as it is being developed for the SIMONA Research Simulator (SRS). In Figure 4 the different modules and their relationships are shown.

On the top of the figure several modules can be seen which generate input of some sort to the system: the Atmosphere module generates atmospheric properties for use in the Aerodynamics and Engine modules. The Pilot Input controls, possibly through an Auto Pilot, the Flight Controls and the Engine, as well as the Aircraft Configuration, e.g. landing gear and flaps. Since these modules are easily exchangeable, a conventional flight control system with simulated pullies and cables, can, for example, be replaced with an advanced fly-by-wire control system without modifications to the rest of the modules.

The middle section of the figure shows the modules responsible for calculating the aircraft's responses: the Ground Interaction module calculates the forces and moments from the landing gear, but also from other ground contact such as a tail or belly scrape. The Aerodynamics module is responsible for calculating the aerodynamic forces on the aircraft, while the Engine module simulates the engines. All the calculated forces are summed and integrated in the Dynamics module, which outputs the state of the simulated aircraft.

The bottom portion of the figure shows some of the modules which use the aircraft's state in some way: The Visual Display System presents a view of the outside world to the pilot, the Motion System module drives the 6 degree of freedom hydraulic Motion System of the SRS, and the Avionics modules control the cockpit displays. The Sound module collects data from other modules to generate a realistic sound pattern in the cockpit.

The entire simulation is controlled by the Simulation Control module, which is part of DUSIME, the subject of the next chapter.

Extension of the concept: DUSIME

DUECA provides all the tools for implementation of a distributed calculation process. For the implementation of a simulation on DUECA, a number of basic capabilities of simulations must be implemented in all modules, typically a simulation can be stopped, started, reset to an initial condition, data can be recorded, and "snapshots", (complete descriptions of the simulation state) can be taken and used to jump back to a previous point in the simulation.

For the implementation of these properties, common to most simulations, a layer of functionality can be added to DUECA. Different layer can be added when the basic functionality of DUECA is used in other domains, for example data acquisition and identification. The DUSIME (Delft University SIMulation Environment) addition provides the following:

- A class SimulationModule, that provides the coordination between different modules in a simulation, for the purpose of control of the simulation. A DUSIME module would not inherit from Module (the DUECA module class), but from SimulationModule. A SimulationModule knows only two or three basic states:

  HoldCurrent The current state of the model is maintained

  Advance A time step is made for the model

  CalculateInco An initial calculation is being calculated

- A class HardwareModule, that functions as a base class for modules that interface with a simulator's hardware. A HardwareModule has a simple set of states:

  Down The hardware is driven to or kept in the off/safe state

  Neutral The hardware is driven to or kept in a neutral state

Active The hardware uses the model input to determine its output

State transitions between HardwareModules and SimulationModules are co-ordinated, basically the **Active** mode is only possible when the SimulationModules are in **HoldCurrent** or **Advance**.

The simple, orthogonal design of the state diagrams for DUSIME modules simplifies implementation for the application programmer. More complex actions, such as a reset to the initial state of the simulation, will be performed by a HoldCurrent state in combination with a reload from a snapshot, while the hardware is in Neutral or Safe.

A second part of DUSIME is formed by a number of standard modules that provide the coordination of the state transitions, and a user interface for the experiment or simulation controller.

Conclusions

At the SIMONA institute there is a need for a middleware layer that Would facilitate the development of simulations and experiments for a research simulator. Current architectures for the implementation of real-time calculation and communication processes can only be used by experts in real-time programming, and would require too much expertise from SIMONA users. A new design, DUECA, was developed. DUECA combines a publish and subscribe mechanism and message passing with with some novel elements, namely (1) explicit allocation of process activation and (2) the conditions under which that allocation should take place, (3) synchronization of data from different sources, and (4) a mechanism to transparently combine processes running at different update rates.

The implementation of an aircraft simulation in this framework shows that the programming model differs from that used in traditional approaches. Activation of modules, and the flow of control in the program, is determined by the flow of data. The composition of a simulation or experiment from modules, and the extension of a program is however greatly simplified by this mechanism. Not only is the data access facilitated by the publish-subscribe mechanism, timing of activation is also provided by DUECA.

An extension to DUECA, DUSIME, further facilitates the implementation of simulation-specific functionalities.

Acknowledgements

The concept of time specification, the ticker and the manner in which Activity objects are used as a metaphor for thread-of-control in DUECA were born and further developed in discussions between the first and third authors, during a stay of the latter at the Delft University of Technology. This stay was supported by a grant from the Van Gogh Foundation, for which we are truly grateful.

References

[1]Anon. EuroSim mk1 - software user manual. Technical report, Fokker Space B.V., Leiden, The Netherlands, 1997.

[2]Anon. Dmso high level architecture. Webpage, http://hla.dmso.mil/, 2000.

[3]Anon. Www homepage: Qnx software systems ltd. Webpage, http://www.qnx.com/, 2000.

[4]R. Kelsey, W. Clinger, J. Rees, H. Abelson, R. K. Dybvig, C. T. Haynes, G. J. Rozas, N. I. Adams IV, D. P. Friedman, E. Kohlbecker, G. L. Steele JR., D. H. Bartley, R. Halstead, D. Oxley, G. J. Sussman, G. Brooks, C. Hanson, K. M. Pitman, and M. Wand. *Revised(5) Report on the Algorithmic Language Scheme*, 1998.

[5]D. Schmidt. Tao overview. http://www.cs.wustl.edu/ schmidt/TAO-intro.html, Oktober 1999.

[6]M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline.* Prentice Hall, 1996.

This article has been cited by:

1. Simon Vrouwenvelder, Ferdinand Postema, Daan M. Pool. Measuring the Drag Latency of Touchscreen Displays for Human-in-the-Loop Simulator Experiments . [Abstract] [PDF] [PDF Plus]
2. Christian Weiser, Daniel Ossmann, Gertjan Looye. 2020. Design and flight test of a linear parameter varying flight controller. *CEAS Aeronautical Journal* **11**:4, 955-969. [Crossref]
3. Pepijn A. Scholten, Marinus M. van Paassen, Max Mulder. A Variable Stability In-Flight Simulation System using Incremental Non-Linear Dynamic Inversion . [Abstract] [PDF] [PDF Plus]
4. Gijs de Rooij, Dirk Van Baelen, Clark Borst, Marinus M. van Paassen, Max Mulder. Supplementing Haptic Feedback Through the Visual Display of Flight Envelope Boundaries . [Abstract] [PDF] [PDF Plus]
5. Tijmen Pollack, Gertjan Looye, Frans Van der Linden. Design and flight testing of flight control laws integrating incremental nonlinear dynamic inversion and servo current control . [Citation] [PDF] [PDF Plus]
6. Thomas Lombaerts, Gertjan Looye, Joost Ellerbroek, Mitchell Rodriguez y Martin. 2017. Design and Piloted Simulator Evaluation of Adaptive Safe Flight Envelope Protection Algorithm. *Journal of Guidance, Control, and Dynamics* **40**:8, 1902-1924. [Abstract] [Full Text] [PDF] [PDF Plus]
7. I. Miletović, D.M. Pool, O. Stroosma, M.M. van Paassen, Q.P. Chu. 2017. Improved Stewart platform state estimation using inertial and actuator position measurements. *Control Engineering Practice* **62**, 102-115. [Crossref]
8. Thomas Lombaerts, Gertjan Looye, Joost Ellerbroek, Mitchell Rodriguez y Martin. Design and Piloted Simulator Evaluation of Adaptive Safe Flight Envelope Protection Algorithm . [Citation] [PDF] [PDF Plus]
9. Dennis Tang, Daan M. Pool, Olaf Stroosma, Q Ping Chu, Coen C. de Visser. Piloted Simulator Evaluation of a Model-Independent Fault-Tolerant Flight Control System . [Citation] [PDF] [PDF Plus]
10. Daan W. J. Van Der Wiel, Marinus M. van Paassen, Mark Mulder, Max Mulder, David A. Abbink. Driver Adaptation to Driving Speed and Road Width: Exploring Parameters for Designing Adaptive Haptic Shared Control 3060-3065. [Crossref]
11. Luisa dos Santos Buinhas, Bruno Jorge Correia Grácio, Ana Rita Valente Pais, Marinus van Paassen, Max Mulder. Modeling Coherence Zones in Flight Simulation During Yaw Motion . [Citation] [PDF] [PDF Plus]
12. N. Beckers, D. Pool, A.R. Valente Pais, M.M. van Paassen, M. Mulder. Perception and Behavioral Phase Coherence Zones in Passive and Active Control Tasks in Yaw . [Citation] [PDF] [PDF Plus]
13. Richard Sopjes, Paul de Jong, Clark Borst, René van Paassen, Max Mulder. Continuous Descent Approaches with Variable Flight-Path Angles under Time Constraints . [Citation] [PDF] [PDF Plus]
14. Ana Rita Valente Pais, M. M. (René) Van Paassen, Max Mulder, Mark Wentick. 2010. Perception Coherence Zones in Flight Simulation. *Journal of Aircraft* **47**:6, 2039-2048. [Citation] [PDF] [PDF Plus]
15. Herman Damveld, David Abbink, Mark Mulder, Max Mulder, Marinus (René) Van Paassen, Frans Van der Helm, R.J. Hosman. Identification of the Feedback Components of the Neuromuscular System in a Pitch Control Task . [Citation] [PDF] [PDF Plus]
16. Mark van den Hoven, Paul de Jong, Clark Borst, Max Mulder, Marinus van Paassen. Investigation of Energy Management during Approach - Evaluating the Total Energy-Based Perspective Flight-Path Display . [Citation] [PDF] [PDF Plus]
17. Thomas Lombaerts, Hafid Smaili, Olaf Stroosma, Ping Chu, Jan Albert Mulder, Diederick Joosten. Piloted Simulator Evaluation Results of New Fault-Tolerant Flight Control Algorithm . [Citation] [PDF] [PDF Plus]
18. Frank Nieuwenhuizen, Marinus van Paassen, Max Mulder, Heinrich Bülthoff. Implementation and Validation of a Model of the MPI Stewart Platform . [Citation] [PDF] [PDF Plus]
19. Redouane Hallouzi, Michel Verhaegen. Subspace Predictive Control Applied to Fault-Tolerant Control 293-317. [Crossref]
20. Olaf Stroosma, Thomas Lombaerts, Hafid Smaili, Mark Mulder. Real-Time Assessment and Piloted Evaluation of Fault Tolerant Flight Control Designs in the SIMONA Research Flight Simulator 451-475. [Crossref]
21. Herman Damveld, David Abbink, Mark Mulder, Max Mulder, Marinus van Paassen, Frans van der Helm, Ruud Hosman. Measuring the Contribution of the Neuromuscular System During a Pitch Control Task . [Citation] [PDF] [PDF Plus]
22. Bram Masselink, Alexander in 't Veld, Max Mulder, René van Paassen. Design and Evaluation of a Flight Director for Zero and Partial Gravity Flight . [Citation] [PDF] [PDF Plus]
23. Peter Zaal, Daan Pool, Alexander in 't Veld, Ferdinand Postema, Max Mulder, Marinus van Paassen, Jan Mulder. Design and Certification of a Fly-by-Wire System with Minimal Impact on the Original Flight Controls . [Citation] [PDF] [PDF Plus]
24. Peter Zaal, Daan Pool, Max Mulder, Marinus van Paassen, Jan Mulder. Multimodal Pilot Model Identification in Real Flight . [Citation] [PDF] [PDF Plus]
25. Jan Comans, Olaf Stroosma, Rene van Paassen, Max Mulder. Optimizing the Simona Research Simulator Visual Display System . [Citation] [PDF] [PDF Plus]

26. Olaf Stroosma, Hafid Smaïli, Thomas Lombaerts, Bob Mulder. Piloted Simulator Evaluation of New Fault-Tolerant Flight Control Algorithms for Reconstructed Accident Scenarios . [Citation] [PDF] [PDF Plus]
27. Arjan Van den Heuvel, M Mulder, M Van Paassen, A In t Veld. Design of a Partial Gravity Flight Director . [Citation] [PDF] [PDF Plus]
28. O. Stroosma, M. van Paassen, M. Mulder, F. Postema. Measuring Time Delays in Simulator Displays . [Citation] [PDF] [PDF Plus]
29. Wim Van Hoydonck, Marilena Pavel. Development of a Modular, Generic Helicopter Flight Dynamics Model for Real-Time Simulation . [Citation] [PDF] [PDF Plus]
30. Walter Berkouwer, Olaf Stroosma, René van Paassen, Max Mulder, Bob Mulder. Measuring the Performance of the SIMONA Research Simulator's Motion System . [Citation] [PDF] [PDF Plus]
31. Edmund Field, Thomas Pinney, Rene van Paassen, Olaf Stroosma, Robert Rivers. Effects of Implementation Variations on the Results of Piloted Simulator Handling Qualities Evaluations . [Citation] [PDF] [PDF Plus]
32. Edmund Field, Rene van Paassen, Olaf Stroosma, Paul Salchak. Validation of Simulation Models for Piloted Handling Qualities Evaluations . [Citation] [PDF] [PDF Plus]
33. Michiel Selier. FLYCAM . [Citation] [PDF] [PDF Plus]