INTEGRATED FLIGHT CONTROL LAW DESIGN, ONBOARD CODE GENERATION AND TESTING FOR SAFETY CRITICAL APPLICATIONS

K.M. Wanie, H. Nier, H. Schmid
European Aeronautic Defence and Space Company, Munich, Germany

To date the development of flight control laws especially for a safety critical environment comprises a lengthy and cost-intensive process.

Functional control law (C/L) design is usually performed applying commercial of the shelf packages (e.g. MATLAB, MATRIXx) and supporting proprietary tools. The specification of the control laws is then passed to the software (S/W) team, which performs hand coding of the onboard software. Finally, specification and code are delivered to the testing team for unit and integration tests.

There are several deficiencies inherent in the process applied today. C/L design is performed quite independently from S/W development. Often the control law specification is delivered in a format (e.g. documents, files), that does not support coding and test as good as possible. For this reason coding, test script development and test execution often require extraordinary effort and a lot of work has to be done manually. Furthermore, communication and cooperation between the disciplines control law design, coding and testing is hampered by a lack of understanding each other.

The present paper addresses these shortcomings by introducing an integrated process for control law design, coding and testing.

Description of the Current Process

Overview

The process commonly adopted for C/L onboard S/W development can be described by the well-known V-Model. An overview of this process is given in Fig. 1.

Starting from top-level system requirements, the left-hand side of the V-Model

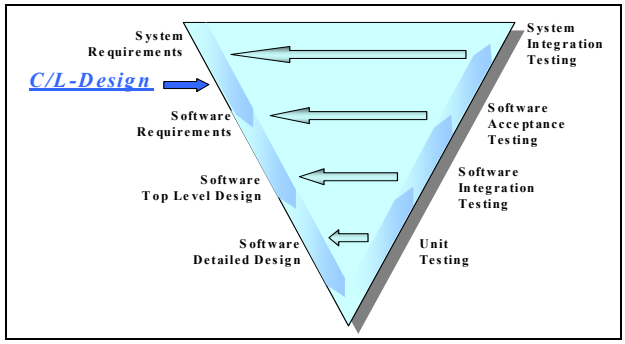

Fig. 1. V-Model for C/L software development

represents the derivation of detailed requirements, the software design and the coding.

The right-hand side subsequently deals with the verification of the coded software against the design process. It has to be proven, that all requirements derived during design are fulfilled.

In the context of the present paper three main steps can be distinguished within the overall process of C/L development:

1. C/L design
2. S/W design and coding
3. Testing

C/L Design

From the V-Model point of view C/L design can be looked upon as a part of the process of deriving software requirements from the system requirements (Fig. 1).

Result of the C/L design is the specification of the control laws for the subsequent steps S/W design, coding and test.

This specification contains a detailed description of all control law elements, their execution order, execution frequency, timing requirements etc. and is usually delivered in the format of written documents and data files.

Additional requirements may originate from other sources e.g. from hardware (H/W) implementation, splitting software execution to different processors or allocation of data in dedicated memory regions.

The C/L specification is passed to the S/W team together with the additional requirements and is the basis for the software development.

S/W Design and Coding

During Top Level Design (TLD) the overall S/W architecture is derived based on the software requirements (Fig. 1). The functionality specified is split into separate modules, interfaces are defined, and the scheduling is derived.

During Detailed Design (DD) the algorithms of the specific functions are developed and specified for coding.

In the coding phase the Detailed Design is coded into onboard software modules and the entire S/W is then passed to the test team together with S/W development documentation and C/L specification.

Testing

The test finally is performed bottom-up from Unit Testing to System Integration Testing (Fig. 1). At each level the test has to prove fulfilment of the corresponding requirements derived during C/L and S/W design.

In the Unit Testing the algorithms of isolated S/W modules are verified against the Detailed Design. Primary goal at this step is to guarantee failure free algorithms by detecting e.g. programming bugs or potential division by zero. To cope with unforeseen inputs also “off-design”-conditions have to be examined.

After the test of the separated units their interaction is verified during Software Integration Test and the cooperation with the system is finally investigated by System Integration Tests.

Test Environment

For the tests described above the separated units have to be stimulated by clinical inputs or by appropriate inputs generated from the C/L model. The output of these test runs has to be compared to the reference output generated on the basis of analytical consideration or simulation runs.

To enable this, a test environment has to be established, which allows reading input data, executing the units under test conditions and comparing outputs with expected results.

This test environment and the test procedures are nowadays derived mainly manually from both the S/W development documentation and the C/L specification with support of dedicated tools.

Representation of Control Laws and Plant

Different types of control law representations and plant models are employed according to the progress of the development process (Fig. 2).

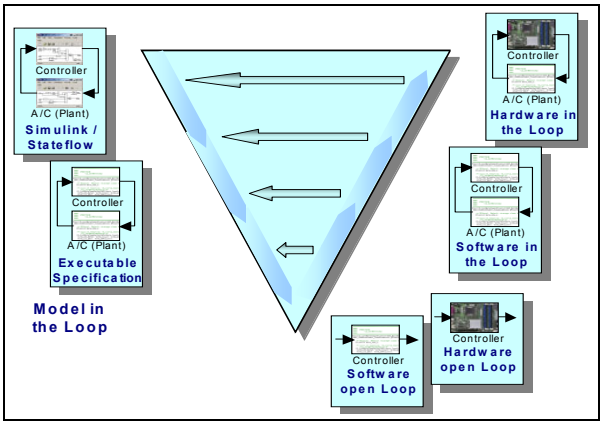

Fig. 2. Representation of control laws and plant

Model in the Loop Simulation

During C/L design a modeling of the control laws is required for design tasks like closed-loop simulation (Fig. 2). Closed loop simulation at this stage is commonly referred to as Model in the Loop Simulation (MILS), since a C/L model instead of the final onboard code is used.

Generally there are linear and nonlinear design tasks and accordingly linear and nonlinear C/L models are involved.

Linear models are required for stability analysis and calculation of linear handling characteristics. They are typically used for tasks like gain- and filter-design and can be derived from non-linear models by automatic or manual linearization. Since there is no direct link to SW development and test, linear models are not further considered here.

The typical proceeding with regard to nonlinear models today is, to realize the control laws as an executable S/W code, commonly referred to as an 'Executable Specification'.

It is important to note that this Executable Specification nowadays during development of complex aircraft usually is totally different from the final onboard code. It may e.g. be written in a different language or have a different S/W architecture.

The reason for these differences is that requirements from target hardware implementation (e.g. certain language restrictions, communication between processors, real time requirements) would hamper C/L design. It should be noted, however, that this hindrance is more and more reduced with evolving technology.

As a second possibility, the non-linear C/L model can be realized as a block diagram in a MATLAB- or MATRIXx-like environment. The problem herewith is, that the modelling of the entire nonlinear behaviour of e.g. a modern fighter aircraft flight control system is extremely complex. Therefore this procedure was not possible for projects in the past, however, gets feasible now, with evolving and improving design tools [1].

Software in the Loop Simulation

During S/W design and coding the C/L model is mapped to the C/L onboard code. During the subsequent test phase the onboard code is examined on both the host and the target computer (Fig. 2).

Simulation runs on the host computer using the onboard software are commonly referred to as Software in the Loop Simulation (SILS). It should be emphasised that in the current context SILS explicitly means usage of final onboard software modules, not of the software described as Executable Specification above.

For the SILS it may be difficult or even impossible to use the complete onboard code as it is loaded onto the aircraft. This is mainly due to hardware specific features like e.g. assembler routines, communication between processors, data exchange via bus systems or dedicated memory access. To achieve a fully 'realistic behavior' of the SILS, a modeling of these hardware aspects is possible. If it is worthwhile to spend this effort has to be decided case by case.

Hardware in the Loop Simulation

The Hardware in the Loop Simulation (HILS) is the most elaborate simulation. The onboard S/W in its final form is loaded into the flight control computer and connected to the plant model.

Modeling of the Plant

The aircraft plant is typically modelled either as a block diagram in a Simulink-like environment or as executable software (Fig. 2).

The later approach is preferable especially with regard to computational performance of the simulation at later stages of the development. It is, however, also possible to create the executable software via auto-coding from an initial Simulink-type plant model.

Especially during HILS test often a part of the plant is represented by dedicated computer hardware boards or by real subsystems (e.g. actuators, sensors).

Finally, the highest degree of modeling (ahead of on-A/C tests) is an iron bird, where real subsystems are used as far as possible.

Deficiencies of the Current Process and Possible Improvements

The main deficiency of the current process is the fact that C/L design usually is performed quite independent from subsequent S/W development and testing. This gap causes a number of problems.

Effort for S/W Design and Coding

As noted above, the control law model used for design usually does not account for target software requirements. An executable specification may e.g. be written in a different language or have a different S/W architecture. In Simulink-type models most of the hardware implementation aspects are usually neglected.

For this reason software top-level design, detailed design and coding on the basis of the C/L specification requires a high manual effort. In addition functional differences may occur between C/L model and target code, which then have to be investigated in depth.

Ideally, manual coding or recoding of the onboard software should be avoided at all. In case of a Simulink-type C/L specification, the onboard code should be derived automatically (by auto-code generation) from the block diagram. In case of an executable specification, the code used during C/L design should be identical (or as close as possible) to the onboard code. If (in the ideal case) the onboard code is developed and used already during C/L design, expensive S/W redesign and recoding as well as implementation differences would be avoided.

Effort for Test Execution

The differences between C/L model and C/L code in the current process cause problems in test as well. Verification may be difficult due to even slightly different behaviour caused by the different architecture or algorithms.

Effort for Test Setup

A further source for considerable effort with regard to time and cost is the setup of test environment, procedures and data, most of which is usually done manually.

Ideally again, test environment, test requirements and data should be derived automatically to avoid manual effort.

Possible Improvements

Main driver enabling the improvements described is the incorporation of requirements from code generation and testing already during control law development. In order to achieve this, an integrated process incorporating a high degree of concurrent engineering is mandatory.

One possible solution is the approach described in the following.

Description of the Integrated Process

Process Overview

A general overview of the Integrated Process is given in Fig. 3.

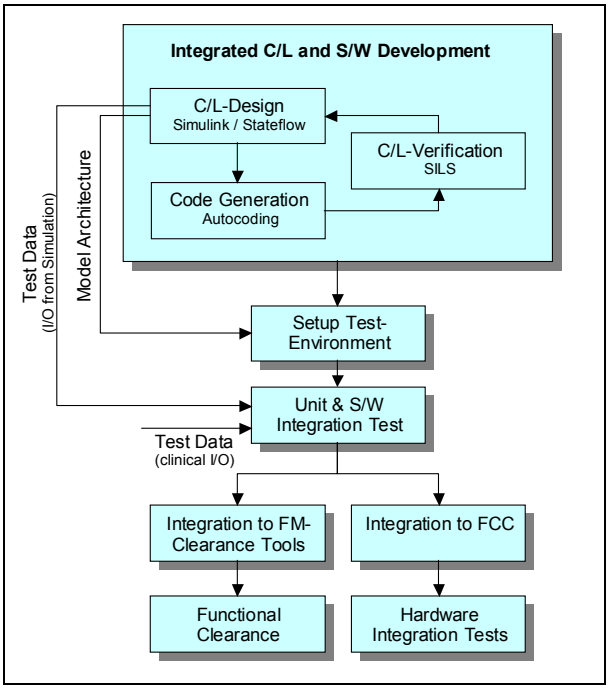

Fig. 3. Integrated C/L development process

The Integrated Process comprises the following steps (Fig. 3):

1. Control law design in the MATLAB / Simulink / Stateflow environment
2. Automated onboard code generation from the Simulink control law model
3. Functional verification of the C/L design by (onboard) S/W in the loop simulation
4. Automated setup of the test environment on the basis of architectural information derived from the Simulink model
5. Automated unit and software integration tests using test procedures and test data derived from the Simulink model
6. Integration of the onboard code into the flight dynamics clearance environment (linear analysis tools, off-line simulation, manned simulator)
7. Functional clearance of the onboard code
8. Integration of the C/L software into the Flight Control Computer
9. Hardware integration tests

Key Features of the Integrated Process

One key issue of this integrated process is the implementation of a single-sourced model based approach. All information required for code generation, test setup and test execution is derived from the Simulink/Stateflow model used during C/L design.

Another key issue is the use of the onboard software modules already during functional design and testing, see Fig. 3.

The C/L and S/W development process is actually represented by steps 1, 2 and 3. They are executed in an integrated design loop, which has several advantages:

- The target code is inherently tested at a very early stage of the development process, resulting in high maturity when delivered to formal software testing
- The requirements from the C/L design and from the H/W implementation point of view are fulfilled inherently
- SILS shows considerable higher performance than MILS using the Simulink model
- For real-time piloted simulation during C/L design an executable code of the control laws is required, which is available with the present approach at no extra cost

Requirements on C/L Model and Code Generator

In order to enable the integrated process a number of requirements have to be fulfilled by both the C/L model and the code generator. Overall target is, to get absolutely identical behaviour (to machine accuracy) of the C/L model and the onboard code.

To achieve this, the C/L model must, of course, consider the capabilities of the code generator. For example, only types of vari-

ables supported by the coder may be used in the model.

The coder on the other hand has to ensure that e.g. execution order and algorithms of the C/L elements or accuracy of variables are exactly reproduced in the code. An important feature in this context is the split of functionality to different functions. This split is actually defined during C/L design by grouping C/L elements into 'Atomic Subsystems' and has to be realized in the code accordingly.

A further important requirement is the capability of incremental code generation, which allows separate coding of changed subsystems. In this way a limited impact of local model modifications on the generated code and the test procedures is achieved and retesting of the entire software in case of small C/L changes is avoided.

Key Features of the Test Environment

To enable unit and S/W integration testing a test framework with capability to read input test data, to execute the units under test and to compare outputs with expected results is required. This framework is generated automatically from architectural information extracted from the Simulink C/L model. This information contains for example the specification of the separate functions (name, I/O parameters) as defined by atomic units in the C/L model and their calling tree.

Supporting Tools

A framework of tools ensuring safety and traceability supports the entire process. Key features are:

- A high degree of automation to reduce the probability of human error
- A configuration management tool tracing and documenting changes in S/W modules, test procedures and test results
- Automated incremental build and test procedures invoking compilation and test only for changed modules

Results

Several code generators have been tested with regard to the integrated process, each of them having particular advantages. In [2, 3] the capabilities of the MATRIXx Ada-Coder have been investigated. Experience is also available for the MATLAB C-Coder and the Ada-Coder (which was available in elder MATLAB versions). ACID4M, an in-house coder mainly for Stateflow applications was developed and tested in [4]. Currently the dSPACE TargetLink C-Coder for MATLAB is under investigation.

As a first application of the integrated process a controller for an air turbine starter motor (ATSM) was developed in Simulink / Stateflow (Fig. 4) and the Stateflow part was coded in C with ACID4M.

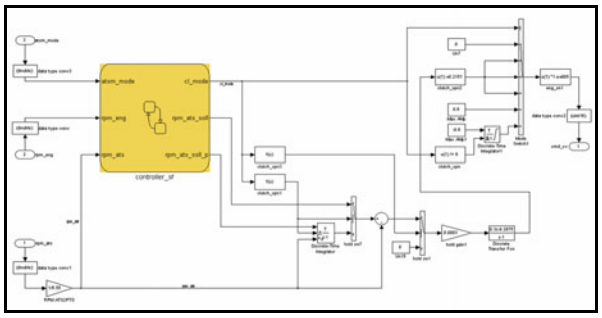

Fig. 4. ATSM-Controller

Fig. 5 shows the comparison of results obtained with the C/L-model and the C/L code. Detailed analysis of the results shows that agreement to machine accuracy is achieved.

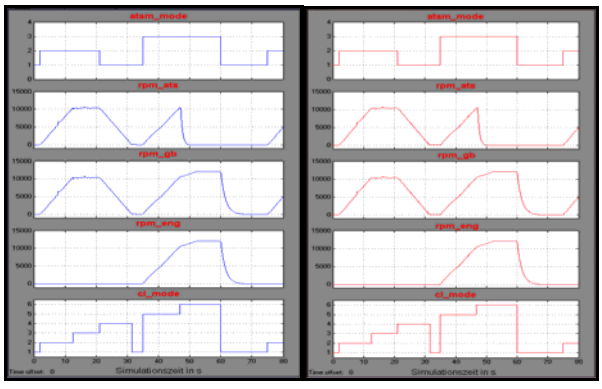

Fig. 5. ATSM results: MILS (left) and SILS (right)

Fig. 6. Flight path simulations

As a second example a part of the autopilot software of a modern fighter aircraft was modeled in Simulink/Stateflow, coded with ACID4M and integrated into the original software. Simulations within flight dynamic clearance environment plotted on top of each other (Fig. 6) again show perfect match of the results from the generated code and the original software.

Conclusions

The experience gained so far with the integrated process shows potential for considerable improvements in terms of development time and cost for flight control law development.

Main driver enabling the improvements is a high degree of concurrent engineering of the disciplines C/L design, S/W development and test

References

[1] Sonnleitner, A. Implementierung und Verifizierung eines Flugregelungssystems für ein modernes Kampfflugzeug. Munich University of Applied Sciences, Diploma Thesis, March 2002.

[2] Faleiro, L. Ada code generation with MATRIXx Systembuild and Autocode in the flight control system design process. Deutsches Zentrum für Luft- und Raumfahrt e.V., Oberpfaffenhofen, May 1998.

[3] Faleiro, L. Current issues in the automatic generation of software code in the flight control system design process. Deutsches Zentrum für Luft- und Raumfahrt e.V., Oberpfaffenhofen, February 1999.

[4] Tas, H. Definition und Erprobung von Komponenten eines modellbasierten Regler-Entwicklungsprozesses. Munich University of Applied Sciences, Diploma Thesis, October 2003.