Nationaal Lucht- en Ruimtevaartlaboratorium
National Aerospace Laboratory NLR

NLR-TP-98599

Integrated management of design processes and data: the CACE environment prototype

J.B.R.M. Spee and D.J.A. Bijwaard

Nationaal Lucht- en Ruimtevaartlaboratorium
National Aerospace Laboratory NLR

NLR-TP-98599

Integrated management of design processes and data: the CACE environment prototype

J.B.R.M. Spee and D.J.A. Bijwaard

This report is based on a presentation held at the 7th Symposium on Computer Aided Control System Design (CACSD'97), [waar ???] en {wanneer ???].

Division:	Flight
Issued:	January 1999
Classification of title:	unclassified

INTEGRATED MANAGEMENT OF DESIGN PROCESSES AND DATA: THE CACE ENVIRONMENT PROTOTYPE

J. Spee *, D. Bijwaard **

National Aerospace Laboratory (NLR), The Netherlands
** Flight Division, spee@nlr.nl ** Informatics Division, bijwaard@nlr.nl*
P.O. Box 90502, 1006 BM AMSTERDAM

Abstract: The drive for efficiency improvement in industry has implications for control engineering also. A process-oriented viewpoint is adopted, highlighting the need for integrated management of design processes and data. The concepts of a design process management system are explained: it has services to support design teams in concurrent design tasks, following a defined flow of activities, using predefined tools on a data repository. The CACE Environment Prototype is presented as an example application in the domain of aircraft control engineering. Most important to efficiency improvement is the automated consistency control of all information in the framework.

Keywords: Computer-aided engineering; computer-aided control systems design; information systems; information technology; systems methodology; project management.

1. INTRODUCTION

Observations of current industry initiatives show a focus on business process improvement. Innovative approaches to processes are recognised as determining factors in competitiveness. The aim is to improve the efficiency in processes; enhancement of product quality, reduction in costs and shorter time-to-market. A representative target in industry is to better efficiency by 20-30 % (O'Toole, 1996).

The need to improve process efficiency also has implications for control engineering activities. Innovative information technology for support of concurrent engineering processes has been applied to the design of control laws for aircraft flight control. The result will be referred to in this paper as the Computer-Aided Control Engineering Environment Prototype (CACE Environment Prototype). The CACE Environment Prototype is the subject of this paper.

NLR employs SPINE for work environments in specific application areas (Baalbergen and Loeve, 1994). SPINE work environments offer transparent use of resources in a network of computers; the 'virtual computing' concept. ISMuS is NLR's control engineering environment based on SPINE, with facilities for management of, and access to tools, data, and documentation (Couwenberg and Cazemier, 1996). The CACE Environment Prototype presented here is a first step towards the integration in SPINE (and ISMuS) of services for design process management.

A process-oriented viewpoint towards engineering environments is adopted in this paper. Efficiency improvement in engineering processes calls for integration of both data management and process management in computer-aided environments. The next section describes the CACE Environment Prototype, illustrating concepts for design process management. Section 3 treats part of a design process for aircraft flight control, and serves as an example for the process dynamics in the CACE Environment Prototype. Section 4 assesses the

significance of having integrated process and data management in engineering environments.

2. THE CACE ENVIRONMENT PROTOTYPE

2.1 Frameworks as the basis for environments

Requirements for support of flight control engineering processes show a lot of commonality with more general requirements for computer-aided engineering. This point has been clearly illustrated by reference to the framework reference model for software engineering (NIST and ECMA, 1993), popularly known as 'the toaster model' (Barker *et al.,* 1993). An open framework approach is regarded as most promising for (control) engineering environments.

Standardisation in frameworks for computer-aided design has been the aim of the CAD Framework Initiative (CFI). The discipline of electronics engineering has been the main driver in the development of an operational framework, compliant with the CFI standards. The result is called SiFrame™ and this product was selected as the core of the CACE Environment Prototype.

Standards and architectures of CAD frameworks are treated by Van der Wolf (Wolf, 1994). He discerns 3 roles in the evolution of CAD frameworks:

- first role - Design Database;
- second role - Design Data Manager;
- third role - Design Process Manager.

The first role of a CAD framework is to act as a common data repository. Tools which share data store it only once, in the design database. Current implementations are: structured organisation of a file system; conventional Data Base Management Systems (management of meta data only, i.e. administrative data, including pointers to actual design data in files); and Object-Oriented Data Base Management Systems.

The second role of CAD frameworks is to provide design management services. The framework exploits knowledge of the structure and status of design information and can therefore provide, for instance: version management; concurrent access mechanisms; organisation of multiple design representations; and query services.

The third role of a CAD framework is to manage design processes. Services for management of design flows (or methodologies) include: control of design activities according to a predefined flow; application of a preselected set of tools; guarantee of the correct data as input and output; and monitoring of the progress in the design process. The characteristics of the CACE Environment Prototype presented here qualify it in this third role; it serves as a **design process management system**.

2.2 Architecture of the engineering environment

The architecture of the CACE Environment Prototype is presented in Fig. 1. It is a framework with integrated tools for the application area. Tools can be integrated as native tools, employing the application programming interfaces to the desktop, database and communication services. Another possibility is to integrate tools as 'black boxes', effectively encapsulating them in command scripts. Encapsulated tools are operated from within the framework, and employ their own user interface and database.

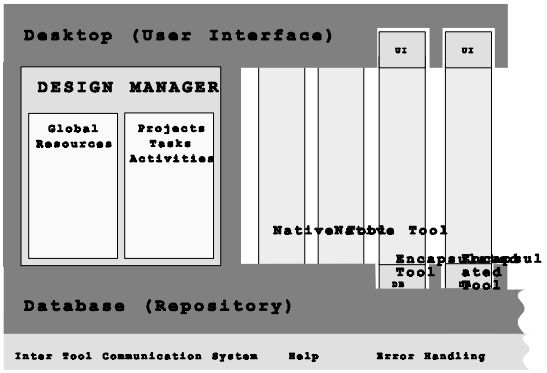

Fig. 1. Architecture of the CACE Environment Prototype

The framework serves as an integration environment for tools and offers common services for the support of design processes. The main framework services in the CACE Environment Prototype are:

- desktop services (user interface);
- database services (repository);
- common basic services; and
- design management services.

The **desktop services** offer access to the design management functions in the framework. All objects in the framework are accessed through the

graphical user interface. Design process objects are presented in trees (hierarchy) or networks. Colours and texts indicate the state of process objects. Changes in process structure or state are reflected immediately on the desktop of all connected clients.

The **database services** are used to manage all design process information in the framework and links to the design results. Each logical database contains information on global resources and multiple design projects. This includes the states of all tasks in running projects. Databases can be physically distributed, to support multi-site development.

The **common basic services** offer general functions to support framework tools. Examples are inter tool communication services, a help system and error handling.

The distinguishing features of this framework lay in the **design management services**. Services for design management hold the core functionality and rules of the framework. One part of design management is concerned with global resources, i.e. the registration, definition and control of resources available to all projects in a particular logical database. The other part of design management involves actual design projects, i.e. the definition of process structure and control of task execution in the framework. The next section describes some of the concepts for design process management.

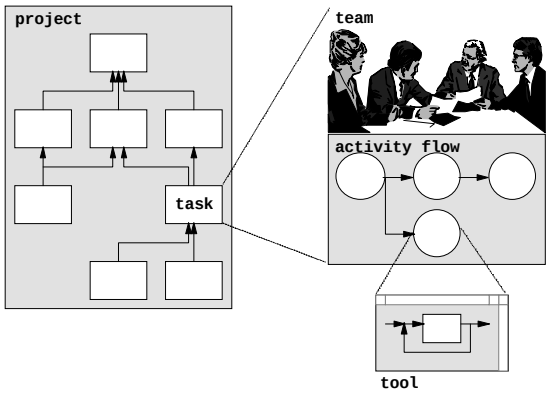

Fig. 2. Process management concepts in the CACE Environment Prototype

2.3 Concepts for design process management

Design process management in the CACE Environment Prototype is centred around the concepts: projects; tasks; teams; and activities in flows (Fig. 2). Whereas these concepts are primarily oriented towards definition, the following concepts support process execution: workspaces; publication; states (behaviour) of tasks and activities; task and activity versions; and consistency.

Projects are the top-level in the design process organisation. The available resources in the project are selected from the global resources. Only members from teams assigned to the project have access to its contents.

Tasks are the building blocks from which a design project is constructed. The task structure resembles the results hierarchy normally defined in projects. At the top is the end result, which is constructed from contributing results in subtasks.

Teams are assigned to each task in the project. Each task has one team attached. Team members can execute design activities, if the task is reserved into a team or user workspace.

Activity flows are an essential element in this framework. The flow defines in which order activities should be executed. As such an activity flow defines the time dependency between activities in one task. More important is the function of guidance to the designers in the team.

Activities are the lowest level at which process execution is defined and controlled. The definition comprises: the tool to be used for the activity; the datatype(s) it can use as input(s); and the datatype(s) it can produce as output(s). Datatypes define data dependencies between activities, both within and across tasks. Fig. 3 gives an example for the activity *Define Controller* in task *Lateral Controller Design*. The names in between parentheses point to the tasks 'producing' or 'consuming' the particular data type.

Workspaces are in this framework to enable concurrency in design processes. Teams and users have to reserve tasks into their workspace prior to performing any activities in them. Reservation of an object locks it from access by others.

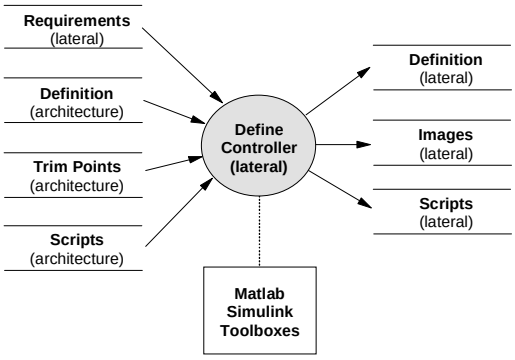

Fig.3. Activity definition with input data types, output data types and tool for execution

Publication is the mechanism to make results available to users outside the workspace in which they were produced. Published data is marked as

read-only in the repository, effectively 'freezing' the design result.

States describe the dynamics of tasks and activities in the design process. A task can be: empty (no previous execution); in work (one/more activities have been executed); partly published (result of activities have been issued); or complete (entire task result has been published). An activity can be: not executed; executed (tool has run with valid result); invalid (input has changed by predecessor activity); or finished (result has been published).

Several **task and activity versions** can be created. This allows for design iterations, with or without storage of design history. Task versions can be linked to create different design alternatives. The framework can show a graph of dependencies between task versions, as a powerful and intuitive means to process monitoring and control.

Consistency is the central and most powerful concept in the CACE Environment Prototype. The framework guarantees a consistent state of all the design activities and the data they produce. Consistency is controlled by application of rules in the framework: before, during and after activity execution; upon publication of results; and upon change of the process structure (task version network). Consistency control is expected to bring the main contribution to efficiency improvement in design processes.

3. AN EXAMPLE: APPLICATION IN A FLIGHT CONTROL LAW DESIGN PROCESS

A design process for aircraft control laws was implemented in the CACE Environment Prototype, including evaluation and documentation of the

design. A part of this process is described here as an example to show the behaviour of the engineering environment.

The task structure is depicted in Fig. 4. Four tasks are in the design project: *Controller Architecture*; *Lateral Controller Design*; *Longitudinal Controller Design*; and *Complete Controller Evaluation*. The *Controller Architecture* task results in the definition of the top level architecture of the controller. The tasks *Longitudinal Controller Design* and *Lateral Controller Design* result in these respective parts of the controller. The combination of both top level architecture and lateral and longitudinal parts of the controller are evaluated in task *Complete Controller Evaluation*.

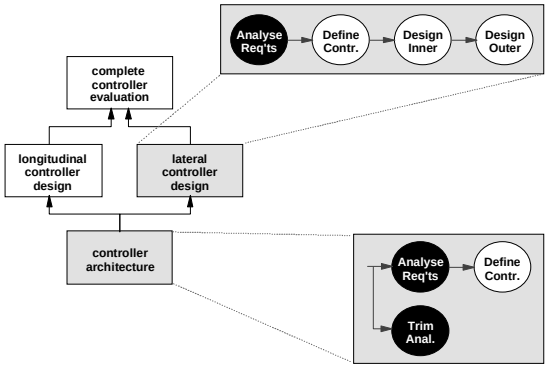

Fig.4. Example design project - task structure and two activity flows

The above results in the following decomposition of tasks. *Complete Controller Evaluation* has (parallel) subtasks *Longitudinal Controller Design* and *Lateral Controller Design*. They have a common subtask *Controller Architecture*. Design results flow from bottom to top in the project.

Two tasks are expanded to show their assigned design activity flows. Tasks and activities are the framework objects in which design teams perform their work. Time dependencies (engineering work flow) are visible in the activity flows. The data dependencies between activities are treated transparently for the designers, but can traced through queries of the database.

3.1 Scenario 1: executing activities.
Execution of a design activity is only possible when the required input data are available. Execution of the activity: starts up the associated tool; reads the appropriate input data from the repository; and locks other users from access. After executing an activity: the tool is closed; the results are written back to the repository; and the state of successor activities is changed to invalid (indicating the need to inspect the impact of modified inputs).

In the following scenario, the effects of executing activities will be discussed. Consider the following situation: the first activities of tasks *Controller Architecture* and *Lateral Controller Design* have been executed (Fig. 4).

When the activity *Define Controller* (in task Lateral Controller Design) is executed, a message pops up that the inputs to this activity are not available. When the database is queried for the activities from which inputs are required it lists *Define Controller Architecture*.

After executing activity *Define Controller Architecture* it is possible to execute *Define (Lateral) Controller* (Fig. 5.). When the database is queried for the activities affected by (re-)execution of activity *Define Controller Architecture*, it will list *Define (Lateral) Controller*. Actual re-execution of *Define Controller Architecture* invalidates *Define (Lateral) Controller* (inspection is necessary).

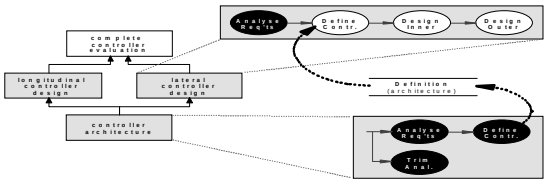

Fig.5. Design project state after execution of activity Define Controller in task Controller Architecture

3.2 Scenario 2: iteration and traceability of results
In the previous scenario, iteration took place by re-executing activities. In this fashion it is not possible to trace back to previous results after re-execution. However, a new task version can be derived from the current task version including its data and connections to the subtasks. The results of this new version can now be altered independent of the task version it was derived from. It is useful to publish a task version before another is derived from it, to maintain full traceability. Looking at the task versions of one task will show the derivation tree (version trace).

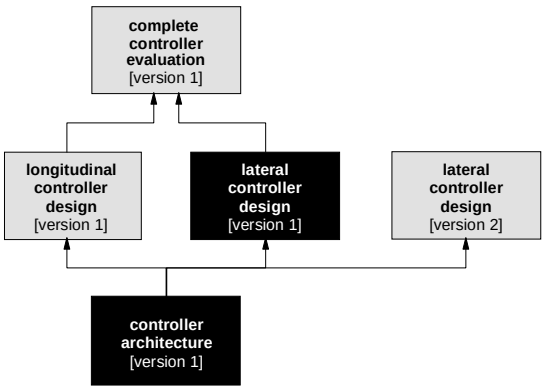

Fig. 6. Initial task versions and task states in the design project

Consider the following situation: all activities in the project have been executed and version 1 of task *Controller Architecture* has been published (Fig. 6). After publishing version 1 of task *Lateral Controller Design*, activities in this task version cannot be executed again. After deriving a new version of this task version, one can however work in the new version. Version 2 of task *Lateral Controller Design* is already connected to version 1 of *Controller Architecture* and contains results, i.e. it inherited the connections to subtasks and the results from the old version (Fig. 7). Once the old task version of *Lateral Controller Design* has been disconnected, the *Evaluate* activity in task version *Complete Controller Evaluation* gets invalidated because the input has been removed. Task version 2 of *Lateral Controller Design* can now be connected to version 1 of task *Evaluate Complete Design*.

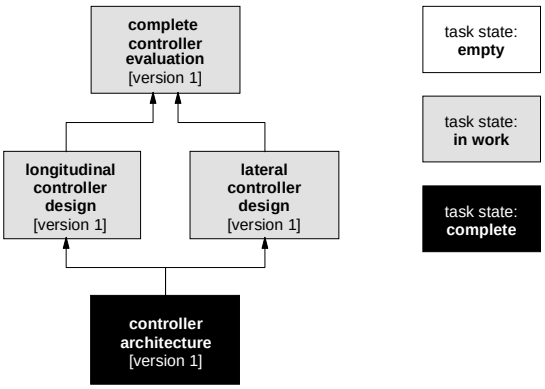

Fig. 7. Task versions and task states after derivation of a new version of task Lateral Controller Design

4. ASSESSMENT

Flight control engineers in aircraft industry indicated a need for better support in data management. Data handling consumes much of a designer's time, because of the large amount of data items involved and their complex relationships. However, data management cannot be viewed separately from process management, because of the number of result versions for each process step and the dependencies between process steps. It is also very difficult to monitor the progress based on data only.

When process management and data management are addressed separately, the users still have to keep the link between the process and the actual design results up-to-date. When a result is ready or has changed, these changes have to be accounted for in the process management package and it is only there that it is visible which other results depend on the new or changed result and have to be checked. So the consistency of the results still has to be checked manually and there is no direct link between the versions of results in version management and the versions of activities in process management.

However when process and data management are integrated with execution of the tools that use and create the data, a much more powerful management is possible. In the CACE Environment Prototype discussed here, there is no longer need to check the link between the different results in the process manually. The framework ensures that the work in the project is consistent at all times by making sure that whenever the result of some activity changes, the right version of results that depend on it have to be checked.

Another important capability is introduced by application of a design management system: the formalisation of design processes. The steps involved and the ordering of those steps can be defined (activity flows in the CACE Environment Prototype). This provides both valuable guidance to design teams and likely efficiency improvements (standardisation).

Storing the design process history is a way to capture design methodologies in design departments ('as is process'). These methodologies can be re-used in subsequent projects. Another possibility is to introduce new processes, by assigning new design activity flows to tasks in design projects and executing them accordingly. Neither approach is trivial though. Important steps are to take a process-oriented view towards engineering environments and to develop a common terminology to discuss process issues.

5. CONCLUSIONS

Integrated management of design processes (including design flows and tool execution) and data provides a guarantee for consistency of the states of all design activities and the data they produce. This reduces the workload on designers and the number of errors made in manual data management, thereby improving efficiency.

A process-oriented view towards engineering environments is necessary. Design processes can be captured, re-used and new design methods can be introduced on the basis of design process management systems. Process issues can be discussed on the basis of a common terminology.

6. ACKNOWLEDGEMENTS

This study was partially conducted under a contract awarded by the Netherlands Agency for Aerospace Programs (NIVR). The following organisations provided the platform on which the environment runs: SiFrame; Informix Software; SUN Microsystems; and Cambridge Control. Essential support was given by the NLR CACEE project team and Cranfield University. Industrial review by British Aerospace and SAAB Aircraft is greatly appreciated.

7. REFERENCES

Baalbergen, E.H. and W. Loeve, *SPINE: Software Platform for Computer Supported Co-operative Work in Heterogeneous Computer and Software Environments*, NLR Technical Publication TP 94359, August 1994.

Barker, A.B., M. Chen, P.W. Grant, C.P. Jobling and Townsend, P. (1993). Open Arhitecture for Computer Aided Control Engineering. In: *IEEE Control Systems*, April 1993, pp. 17-25.

Couwenberg, M.J.H. and R.J. Cazemier, NLR's CACE working envrionment ISMuS, In*: Proc. IEEE Itn. Symp. on CACSD*, pp. 468-471, Dearborn, 1996.

NIST and ECMA (1993). *Reference Model for Frameworks of Software Engineering Envrionments (Technical Report EcMA TR/55, 3rd Edition, NIST Special Publication 500-211)*, August 1993, U.S. Government Printing Office, Washington.

O'Toole, K. European Aerospace Survey 1996. *Flight International*, 4-10 September 1996, 49-51.

Wolf, P. v.d. (1994). *CAD Frameworks, Principles and Architecture*, Kluwer Academic Publishers, Dordrecht, the Netherlands.