



NLR-TP-2005-066

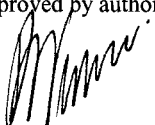
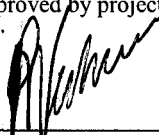
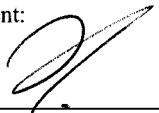
Prototyping interactive cockpit applications

R.P.M. Verhoeven and A.J.C. de Reus

This report has been based on a paper presented at the 23rd Digital Avionics System Conference, Salt Lake City, Utah, U.S.A., October 24-29, 2004.

This report may be cited on condition that full credit is given to NLR and the authors.

Customer: National Aerospace Laboratory NLR
Working Plan number: AT.1.G.3
Owner: National Aerospace Laboratory NLR
Division: Air Transport
Distribution: Unlimited
Classification title: Unclassified
February 2005

Approved by author: 	Approved by project manager: 	Approved by project managing department: 
--	---	---



Summary

In the early phases of a cockpit application development process prototypes are built for initial evaluation and demonstration purposes. It is in this phase that innovative ideas are made more concrete, and a base for the specification is created. The prototypes are improved by performing several design-review cycles. The iterative nature of prototyping demands a flexible and customizable design and evaluation environment.

Using the experience obtained in a variety of research projects during the last decade, NLR has developed a prototyping environment that facilitates the rapid development and evaluation of avionics display formats. It supports a component based design strategy, enabling efficient re-use of previous work and facilitating parallel activities of several project team members. The resulting display formats can be rapidly deployed to the appropriate evaluation platform, ranging from desktop test environments, via cockpit mock-ups, up to high-fidelity flight simulator and even flight test aircraft.

The tools have been successfully applied in several cockpit application development projects. They have allowed to drastically reduce the effort and duration involved with implementing the prototypes, so that customer research issues can be addressed earlier and in a more cost-efficient way. Furthermore, the tools enable evaluated prototypes to be used as a base for further development towards a certified cockpit application running on an avionics target system.



Contents

1	Introduction	4
1.1	Background	4
1.2	Cockpit applications	4
2	Design methodology	5
2.1	Product development processes	5
2.2	Research processes	6
2.3	Supporting tool	6
3	Flexible prototyping	8
3.1	User requirements	8
3.2	Interpreter-based viewer	8
3.3	Integrated graphical and behavioral prototyping	9
3.4	Hybrid data-driven and event-driven communication	11
3.5	Component-based design strategy	12
4	Realistic evaluations	14
5	Base for further development	14
5.1	Using prototyping results	14
5.2	ARINC 661 compatible	14
6	Example application	15
7	Conclusion	16
	References	16

(16 pages in total)



1 Introduction

1.1 Background

After the transition from electromechanical instruments to the glass cockpit, a new revolution is rapidly taking place on the civil flightdeck. The introduction of interactive cockpit applications allows the flight crew to operate the aircraft systems using a cursor control device in a way that is very similar to operating personal computers. While this potentially allows a more user-friendly working environment for the pilot, it imposes new and demanding challenges to the industry and the aeronautical research community alike.

Various aircraft and avionics manufacturers are currently taking up the task of transforming their design processes that are optimized for one-way information presentation into efficient means to develop interactive cockpits. This includes changes in the interaction between avionics integrators and equipment suppliers.

Manufacturers offering aircraft with truly interactive cockpits to their customers include Airbus (with the A380), Dassault Aviation (Falcon 900EX and 2000EX), and Embraer (170 and 190 series), while Boeing will likely join with the 7E7 and former Fairchild Dornier offered the 728JET until its collapse. Not surprisingly, all major avionics manufacturers offer equipment for interactive cockpits.

1.2 Cockpit applications

In this paper we are using the term cockpit applications instead of cockpit displays or cockpit instruments. We feel that it better expresses the nature of the nowadays “software” flightdeck and the generally accepted view that user-interface and systems are equally important.

In the electromechanical era, the flightdeck featured dedicated instruments for each system. In the current glass cockpit, information is integrated and presented in a graphical way, but essentially it is still one screen per system and dedicated hardware for each system. Modular avionics radically transforms this concept into a network of generic hardware running software applications that are specific for each system. The cockpit display system is part of this network and the pilots are operating software in a way similar to an office worker with a computer connected to a company network.

In a user-centered flightdeck design method, development of the user-interface and of the underlying system is done side-by-side. They are equally important for reaching a system that is effective, efficient and comfortable to use. For instance, in a flight management system (FMS) with a very cumbersome user-interface pilots will most likely not exploit the system’s full potential. On the other hand, an FMS with a very convenient user-interface but lacking important functions will not be very helpful in optimizing the flight.



2 Design methodology

2.1 Product development processes

Design of cockpit applications usually consists of several design-review cycles (Figure 1). These cycles are intended to improve and finally validate the user interface, both in a technical sense and for efficient operation by the pilots. The aim is to get the user interface right first time, so that costly re-engineering after the product is introduced on the market can be avoided. These design-review cycles are formalized by new regulation on human factor aspects of flightdeck design.

The first cycle normally starts with a very simple prototype, or a quick adaptation of an existing design. Such a design could be done on paper or using a computer presentation tool. The review is then performed utilizing the paper or electronic means.

In later cycles the design is gradually improved and completed. The evaluations are also gradually more encompassing or realistic. Naturally the more complete – and often more complex – later cycles are also more costly. This way of working is also known as iterative and incremental development (IID)[1].

By performing evaluations with users early in the process, potential problems can be discovered and solved at a relatively low cost. This way, the number of problems appearing in the later – more costly – cycles can be drastically reduced.

A process such as this can only be cost-effective when the prototypes can be efficiently adapted and extended throughout the complete prototyping phase. Optimally, the best prototype should also be directly usable in specification and design, and for (automatic) coding.

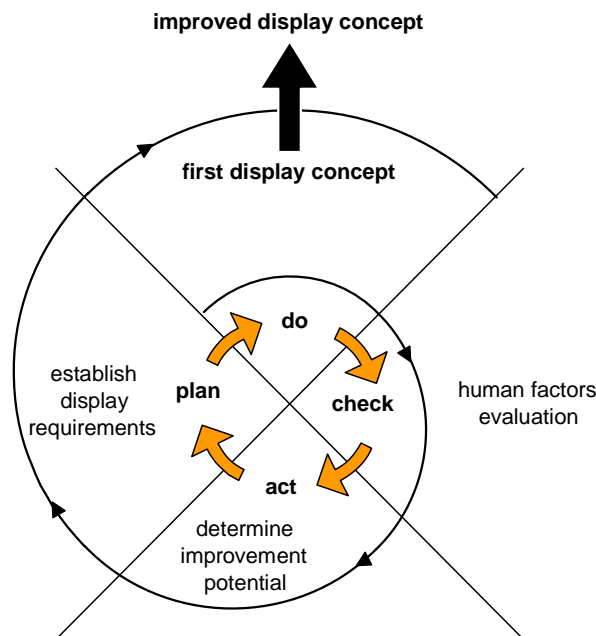


Figure 1. Design-review cycle



2.2 Research processes

Although the objectives and the end product are not the same, in itself the design iterations performed in a – customer focused – research environment are not fundamentally different from those in product development.

Also in a research environment a requirement exists to develop a first prototype of the human-machine interface rapidly, at a low cost and often in a few variations. In an efficient process the most promising HMI “sketches” can then be easily extended and completed for evaluation with pilots in a more realistic environment like a flight simulator, and if needed, used as a base for the specification phase.

For example, at NLR we develop cockpit human-machine interfaces on desktop PC’s and this is also the platform for informal reviews early in the process. We usually perform the first formal evaluations in a cockpit mock-up. For more advanced evaluations and for validations, we can use a full-flight research simulator or one of our aircraft (Figure 2). With a variety of evaluation platforms, it is evident that we have aimed to optimize our process, so that transitions of HMI concepts from one platform to the other require only a minimal effort.

2.3 Supporting tool

To accommodate the prototyping process in a research environment, NLR has developed a prototyping tool - called *Vincent*¹ – which fulfils three important requirements:

- Enable flexible prototyping
- Support realistic evaluation
- Serve as base for further development

The following sections will discuss *Vincent* in more detail with respect to these requirements.

¹ Visual Interface design of New Control concepts for Effective and Natural Task performance, also referring to Vincent van Gogh, Dutch painter, 1853-1890.



1

Desktop Evaluation Environment (“Airsim”)

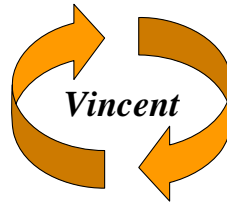
- Displays and control panels on one PC
- Typically early, low-fidelity evaluations
- Review with experts and team members



2

Cockpit Mock-Up (“Apero”)

- Three LCD's, one PC per LCD
- Typically early, medium fidelity evaluations
- Review with experts and test pilots



4

Laboratory Aircraft Cockpit (“PH-LAB”)

- One large LCD at F/O position, one LCD in the back, PC-based
- Typically in-flight validations and demonstrations
- Evaluation by test or airline pilots



3

Full-Flight Research Simulator Cockpit (“Grace”)

- Fully reconfigurable cockpit, displays PC-based
- Typically high-fidelity validations of mature concepts
- Evaluations with airline pilots

Figure 2. Vincent displays in four evaluation environments



3 Flexible prototyping

3.1 User requirements

The prototype phase is an important part of cockpit application development. The flightdeck designer develops ideas and concepts that could meet the requirements and comply with the flightdeck design guide. Usually there is no single solution for the user interface. Ideas and concepts develop over time as insight in the issues grows and as a result of evaluations with test pilots or pilots from the end-user population. Often ideas and concepts are combined to arrive at a more optimal solution. All in all the prototyping phase is a highly creative process, with many informal and formal review moments and consequently frequent changes to the prototypes. Clearly, due to the highly iterative nature of the prototyping phase, it should be easy to build a prototype and straightforward to change it. More specifically, it should be possible to

- build or modify graphical design,
- add or change prototype behavior,
- prototype symbology as well as interactive display applications,
- re-use previous work, and
- compare alternative designs.

During the development of the prototyping tool these requirements were the focus of attention and have resulted in the following key characteristics of *Vincent*:

- The tool is interpreter based.
- The tool allows integrated graphical and behavioral prototyping.
- The tool supports data-driven as well as event-driven communication.
- The tool features an extensible framework of user-defined components.

Two important tools used in the *Vincent* prototyping environment are the *Vincent* editor and the *Vincent* viewer (Figure 3). The editor is used to design prototypes. Both graphical as well as behavioral aspects of a prototype can be addressed within the editor. The viewer is used to evaluate the prototype within the desired simulation environment.

3.2 Interpreter-based viewer

Because the viewer is interpreting the prototype definition as produced by the editor, there is no need for code generation and compilation to run the prototype. This significantly accelerates and simplifies the design-review cycle, and therefore considerably increases the flexibility of the prototyping process.

Interpretation usually comes at a cost: run-time performance is often significantly lower than with code generator based tools. However, the *Vincent* viewer is highly optimized and it is possible to obtain (very) high update rates on relatively modest hardware: a standard desktop

PC with hardware support of OpenGL is sufficient. The result is smooth display of even highly dynamic information and more than sufficient responsiveness to user inputs.

The interpreter-based viewer has proven itself in a variety of cockpit application development projects. Using the viewer, it was also possible to simply exchange dynamic displays with project partners for review, evaluation and demonstration purposes.

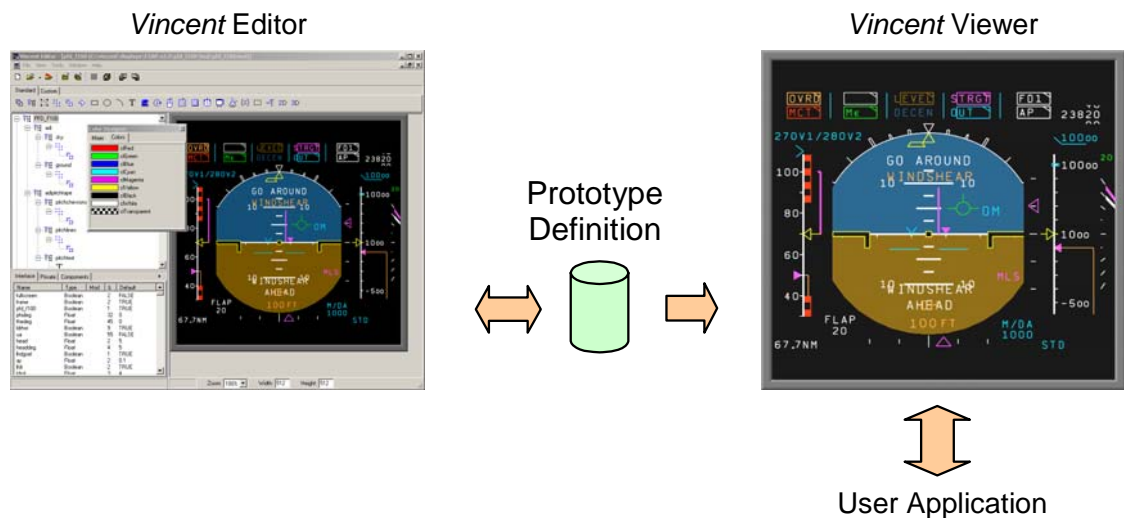


Figure 3. Vincent tool chain with Editor and Viewer

3.3 Integrated graphical and behavioral prototyping

A Vincent display prototype is described by a *hierarchical scene graph* structure. This type of data structure is often used in graphical programming models (i.e. VRML [2]), because it establishes a clean boundary between display representation and display rendering [3].

The scene graph contains *nodes* that describe objects and their properties. Objects may represent a geometrical primitive (e.g. line, polyline, rectangle, ...) which can be visualized. However, they may also represent other functionality, such as a sensor to detect user input, or a camera to control how the graphical objects should be projected on the screen.

Nodes can contain *fields* of various types, dependent of the object property they represent. The fields are the inputs and the outputs of the nodes. Using the input fields the standard behavior of a node can be overruled. Specific fields may contain for example a position, scaling factor or visibility flag. Other specific fields may contain one or more children nodes. With the exception of the topmost root node, every node in the scene graph describing the prototype has one or more parents (directed acyclic graph).



Vincent supports a large variety of built-in nodes, giving the HMI designer the required flexibility. One group of nodes was not mentioned yet: the *traversal control nodes*. An example of a powerful traversal control object is the Loop node. This node has an input field *count* and an output field *i* and its children are traversed *count* times during rendering. Children of the Loop node can refer to *i*, which is incremented after each iteration. This way rendering can be iteration-dependent.

The *transformation hierarchy* includes all nodes and their relationships that are responsible for the graphical part of the application design. The *behavior graph* describes the connections between fields and the flow of events through the prototype. It is common practice during the design to build the transformation hierarchy and the behavior graph simultaneously. In the *Vincent* editor, the transformation hierarchy is built with drag-and drop. Behavior is added by defining the *interface fields* of the prototype and connecting them to the appropriate node fields within the scene graph.

The relationship in between an interface field and a node field may be a direct connection, but it may also involve an *expression*. Expressions are useful because they allow for defining a purely *functional interface*. Such an interface is fully tailored to the underlying application, and independent of the coordinate system in which the graphical objects are defined.

The ability to model the graphical and behavioral part of a prototype within the same environment eases the design and evaluation process significantly. For example, a *Vincent* Primary Flight Display (PFD) prototype (Figure 4) can be fully controllable by functional inputs, such as roll in degrees, pitch in degrees, and altitude in feet.

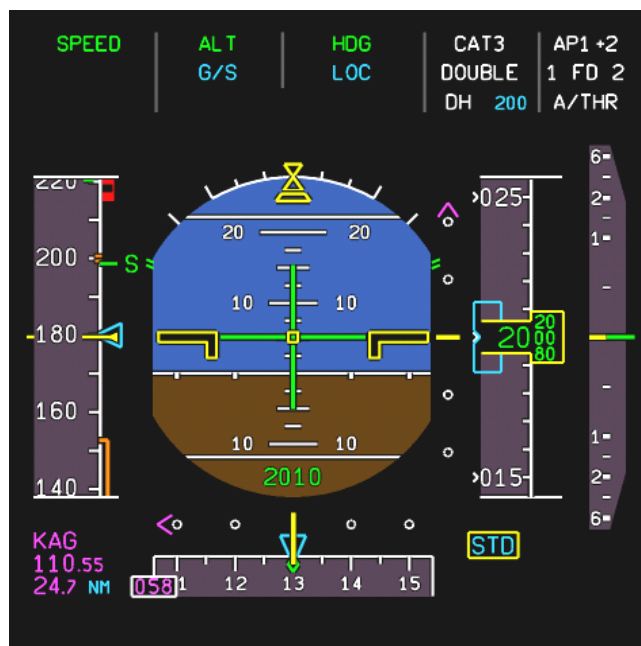


Figure 4. PFD with purely functional interface

3.4 Hybrid data-driven and event-driven communication

Generally, two types of cockpit applications can be distinguished: symbology applications and interactive applications. Symbology applications present information one-way to the pilot. Examples of symbology display applications are the Primary Flight Display and Navigation Display (ND) as can be found on most current flightdecks.

Typically symbology displays are *data-driven* applications. Interactive applications enable the pilot to react on the information shown by means of, for example, a toggle button, push button or combo box. This type of information exchange has an *event-driven* character. Some applications contain both data-driven symbology and event-driven parts. For instance, the navigation symbology in an interactive ND (Figure 5) is data-driven, while it also responds to pilot inputs with the cursor control device.



Figure 5. Interactive ND prototype²

To enable prototyping both types of application, *Vincent* supports a hybrid data-driven and event-driven communication model. Data-driven behavior is modeled using expressions, as discussed in the previous section. Event-driven behavior can be modeled using *scripts*. Each time an event occurs, the associated script will be executed.

Figure 6 (next page) shows a conceptual execution model of the *Vincent* viewer. A design-time scene graph is built by parsing the prototype definition. The design-time graph contains the complete prototype, including the prototype behavioral described by expressions and scripts.

² This display prototype was developed by NLR in the EU-funded MA-AFAS project.

Synchronized with the User Application providing input data, a run-time scene graph is built by evaluating the expressions contained in the design-time scene graph, and rendered to obtain a visualization of the prototype.

The pilot may inject events in the system by moving the cursor or selecting an object. These events are "transmitted" by output event fields of particular nodes. The scripts attached to these fields will be executed a-synchronously, and may cause other scripts to be fired as well. After all scripts have been executed, a new evaluate-render cycle will be made to update the prototype visualization.

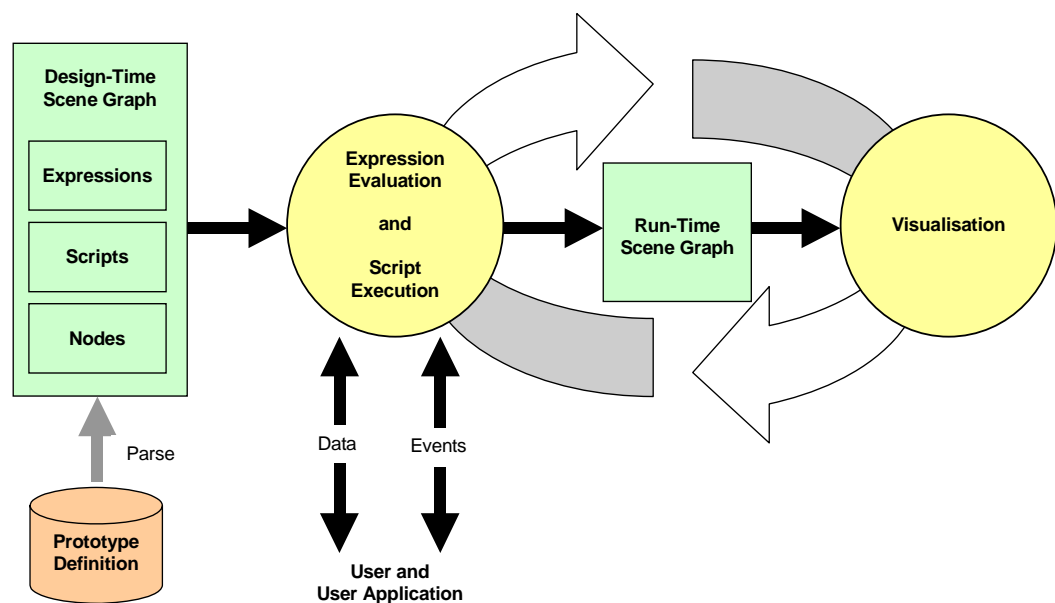


Figure 6 Execution model of the Vincent Viewer

3.5 Component-based design strategy

By allowing for using user-defined nodes besides the built-in nodes in the scene graph, *Vincent* enables a component-based design strategy, which has many advantages. One of their main advantages is that components hide their internal complexity to the user (encapsulation). Only the component interface is relevant to the user of the component.

User-defined components can be of any size from very small to very large, and behave just like built-in nodes. An example of a very small component is a digital readout with a specific formatting according to the value displayed. A somewhat larger example is a complete speed tape. In fact, a display itself can also be used as component in a larger structure, like a complete cockpit layout. *Vincent* components can be made completely self-contained, due to the ability to hold graphics as well as functional behavior.

The ability to use components stand-alone or as part of a bigger structure without any modification, increases the flexibility of the prototyping process significantly. Each display component communicates directly with its own User Application, as if it is running stand-alone (multi-channel communication; see Figure 7 and Figure 8). Optionally, the interface of a sub-component may be extended with interface fields exchanging information with its parent component (in this case the top-level component).

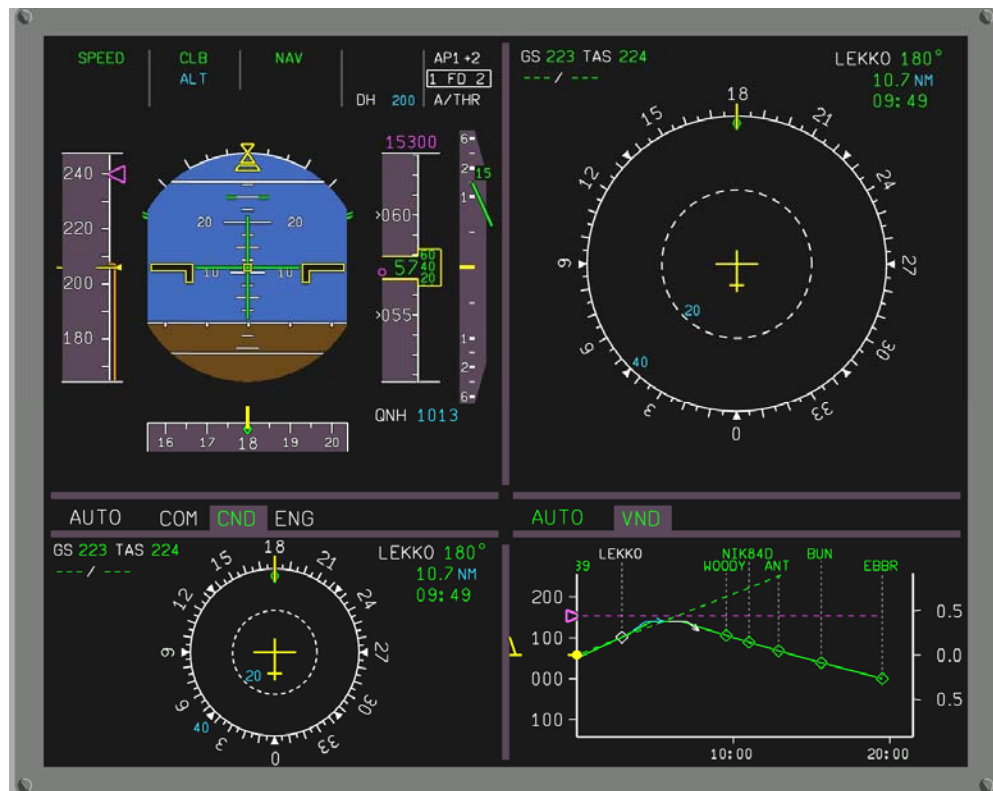


Figure 7 Four displays in a larger structure

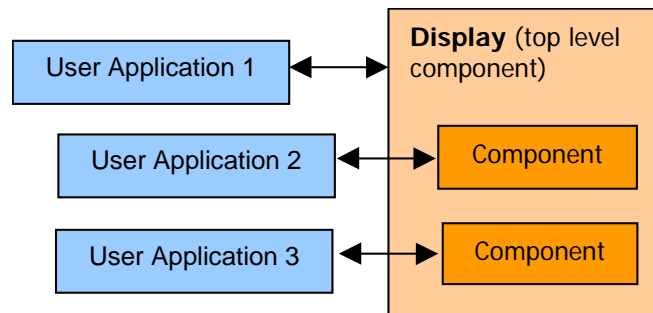


Figure 8. Multi-channel communication



4 Realistic evaluations

The *Vincent* viewer is a high-performance interpreter, built on the advanced graphics library OpenGL. To give an impression of the performance, a fully modeled PFD including behavior and communication (e.g. Figure 4), renders at least with 120 frames per second (fps) with today's PC hardware and graphical board; the display in Figure 7 with 30 fps or more.

These performance characteristics of the interpreter enable prototype evaluation in a realistic simulation environment. Depending on the level of realism required, display prototypes can be implemented on a variety of platforms, ranging from a desktop configuration, up to high fidelity moving base flight simulators, like for example the GRACE simulator at NLR (Figure 2). Note that in case of a desktop configuration, the relevant pilot controls are also simulated using *Vincent*.

5 Base for further development

5.1 Using prototyping results

In a product development process, prototyping results serve as a base for further development. In the IMCAD project [4], it is investigated how to improve the cockpit application development process, primarily focussing on the prototyping and specification phases. Effective use of prototyping results reduces overall development costs, and speed-up time-to-market.

In the context of this project, *Vincent* has been made capable of exporting the prototyping results of symbology displays – in an extensible, human readable XML format – for further use in the development process. Also interactive applications requiring compliance with the new ARINC 661 standard [5] can be prototyped using *Vincent*.

5.2 ARINC 661 compatible

The ARINC 661 standard defines two interfaces in between Cockpit Display System (CDS) and avionics equipment (user system). The first one is the interface between the avionics equipment and the display system graphics generators. The second one is a means by which symbology and its related behavior is defined.

The standard relies on a basic set of graphical user interface objects, the so-called ARINC 661 HMI Widget Library. The standard describes the widgets in a purely functional way, without any reference to a visual representation, allowing any company style. Because some of these widgets enable user input, by means of cursor control devices or keyboards, interactive displays can be specified based upon the ARINC 661 standard.

The capability of *Vincent* to define and instantiate graphical components with encapsulated behavior allows building an ARINC 661 compatible widget library, with a user-defined look and feel. This ARINC 661 compatible widget library can be used to prototype interactive cockpit applications. After design and evaluation, the prototype can be saved in an XML format³, and serve as a base for further development.

6 Example application

Vincent has been used successfully in a variety of flightdeck design and display development projects, both military and civil. One of the projects exploiting much of the potential of *Vincent* is the flightdeck design prototype developed in the NEWSCREEN project [6]. In this interactive concept, component-based design and the capabilities of the multi-channel, interpreter-based viewer were fully exploited. This has dramatically reduced the development duration and effort and has resulted in a favorable concept (Figure 9).



Figure 9. NEWSCREEN flightdeck concept

³ A draft version of this format has been proposed by the ARINC 661 Working Group, and sent to the ARINC committee for approval.

7 Conclusion

The prototyping phase of a cockpit application development process demands a flexible prototyping environment, in which prototypes can be easily adjusted and realistically evaluated in the appropriate simulation environment.

The ability of *Vincent* to prototype graphical and behavioral aspects within a single hierarchical component-based data structure has proven to be very effective. Also the use of an interpreter to evaluate the candidate prototypes has significantly increased the flexibility of the prototyping process. Furthermore the possibility to define hybrid data-driven and event-driven communication has broadened the application domain. Finally, re-use of prototyping results in the next phases of a cockpit application development process, makes the prototyping effort even more efficient.

References

- [1] Mark van Harmelen (editor), Object Modeling and User Interface Design, Addison-Wesley, March 2001, ISBN 0-201-65789-9
- [2] Virtual Reality Modeling Language, VRML, Web3D Consortium, <http://www.web3d.org/>
- [3] Aaron E. Walsh, Understanding Scene Graphs, Dr. Dobb's Journal, July 2002
- [4] Improving the Cockpit Application Development Process, IMCAD project website, <http://www.nlr.nl/public/hosted-sites/imcad>
- [5] Aeronautical Radio Inc, 2003, Arinc specification 661-1, Cockpit Display System Interfaces to User Systems
- [6] Three Large LCD Cockpit, NEWSSCREEN project website, <http://www.nlr.nl/public/hosted-sites/newscreen>

E-mail addresses

Ronald Verhoeven, Training, Human Factors and Cockpit Operations Department, Air Transport Division, NLR, The Netherlands, verhoeve@nlr.nl

Antoine de Reus, Training, Human Factors and Cockpit Operations Department, Air Transport Division, NLR, The Netherlands, dereus@nlr.nl