

# VARIABLE ASPECTS AND MODELS FOR REUSABLE COMPONENT-BASED AVIONICS SYSTEMS<sup>i</sup>

David Sharp  
*david.sharp@boeing.com*

*The Boeing Company  
P.O. Box 516  
St. Louis, MO 63166, USA*

## Introduction

In 1995, an initiative was launched at Boeing (then McDonnell Douglas) to assess the potential for reuse of operational flight program (OFP) software across multiple fighter aircraft platforms, and to define and demonstrate a supporting system architecture based upon open commercial hardware, software, standards and practices.<sup>ii</sup> The following year, this became a key element of the Bold Stroke Open System Architecture avionics affordability initiative which applied these techniques to the broader tactical aircraft mission processing domain.

Key aspects of the component-based logical architecture developed therein have been previously described.<sup>iii</sup> To support the differences between multiple products within the family, variable aspects were separated from the business logic which is the primary reuse focus. This paper describes aspects which were separated, models which supported their later specification, and some key domain requirements. Some defined aspects are not included due to space limitations, as well as definitions of the XML schemas which communicate these aspects between development tools.

The primary goal of separating variable aspects from the elements in the component library is to enable late binding of their values and increase reuse. This deferral of specification increases flexibility but also increases the burden on component integrators, motivating the need for automated tool support. Figure 1 provides a broad outline of the proposed embedded system component integration approach. In the beginning of the process, reusable components are provided as input to product integration. This step provides all associated artifacts such as UML models and source code, and defines the classes that are available for integration. A set of models is then created to specify the different variable aspects of the desired product, namely by binding the variable aspects associated with component instances. Automated configuration tools are used to synthesize configuration code for executable systems satisfying the

requirements. These modeling and generation steps are the focus of this paper. In a full development scenario, analysis, build, and test activities would be included as well.

Essential elements of this approach were developed under the DARPA (Defense Advanced Research Projects Agency) Information Exploitation Office (IXO) Model-Based Integration of Embedded Systems (MoBIES) and Program Composition for Embedded Software (PCES) programs.

## Variable Aspects And Their Models

Perhaps the key architectural challenge in this environment is the separation of variable aspects from the component implementations. In a distributed real-time embedded (DRE) system context, this challenge must be met while still satisfying stringent performance requirements within constrained resources. In an ideal development environment, each variable aspect is specified during development of a specific product through an associated modeling view. Thus variable aspects and modeling views (also known as “subviews”) become synonymous.

A key challenge of multiple view modeling for real-time embedded systems arises from the cross-cutting nature of the different aspects. For example, changing the allocation of components to processes and processors affects scheduling and fault tolerance aspects. The use of multiple views for analysis and automated composition requires their integration to

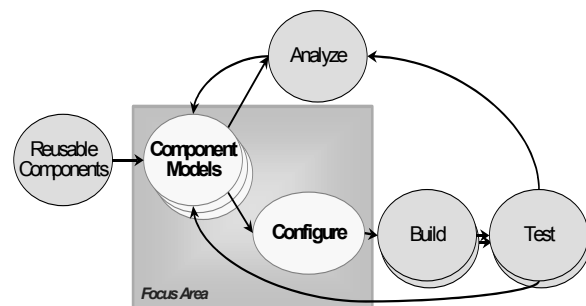


Figure 1: Component Integration Approach

provide a coherent view of the entire system and highlight effects of changes in other views.

Scalability of these approaches is a key concern. The avionics systems targeted by this approach include millions of lines of source code and thousands of components. The size of these systems greatly benefits from automated development aids. Wherever possible, these modeling activities should be possible both manually and via input from configuration automation tools, including monitors of actual execution of partial and complete system configurations.

Since the focus of this paper is on extra-functional performance properties, the Process View and Deployment View are emphasized where extra-functional properties are most visible. The Design View principally is used to model the functional aspects of components and systems from a logical perspective. This is particularly true in product line development environments that are intended to span multiple hardware configurations.

## Process View

Execution control and timing are a critical concern of large real-time embedded systems and in practice are often handled empirically. This has led to substantial rework, cost, and system development delays in many important cases. Improving this current practice is one of the major motivations for moving to the explicit modeling and model-based analysis of such properties. The Process View is used to identify the processes and threads executing in a system and to identify which components are executing in each thread. In the envisioned development approach, it will also be used to identify dependencies between threads, between components, and between threads and components in order to facilitate the desired analysis. This subsection will examine the modeling needs for various subviews within the overall Process View.

## Execution Dependency Aspect

Control flow in avionics systems has a significant impact on, and is impacted by, extra-functional properties and requirements. An activity may be executed in response to new data, at some periodic rate, or in response to a request by a component. The goal of this view is to be able to model the execution dependencies of specific components on specific triggers, or events, as well as the dependencies between components that generate and respond to such triggers.

In some cases, the periodic arrival of data triggers execution of components that receive the data. These components in turn trigger other components, and represent roots of acyclic execution dependency graphs.

Other such graphs may be initiated via time triggers. Within these graphs, subgraphs often exhibit modal behavior and are conditionally active based on the state of the system. Modeling these graphs in a way practical for large systems is a key modeling challenge. Such modeling is ideally performed without reference to a particular hardware configuration. Hardware specifics are used to refine models using the Deployment View.

The following are examples of attributes within this aspect subview:

- Simple trigger types: these could be as simple as names or enumerations that convey the semantics of the trigger. Trigger types could be used to provide filtered views or highlighted relationships within a view.
- Composite triggers: components that respond to Boolean combinations of triggers, or richer combinations, in addition to the occurrence of individual triggers.
- Execution dependency graphs: this includes the specification both of the roots of these graphs, typically triggers related to time-based interrupts, and the arcs within the graphs representing execution dependencies between component instances. Following Deployment View modeling, some of the execution dependencies may result in distributed computation.
- Execution rates for the time-based roots of execution dependency graphs.

## Invocation Dependency Aspect

The subject architecture leverages many of the concepts in the CORBA Component Model while retaining the performance required for DRE systems.<sup>iv</sup> The Invocation Dependency subview captures the relationships between receptacles (i.e. required interfaces from a particular component) and facets (i.e. server component implementation interfaces).

This subview allows users to:

- Specify invocation relationships between client components and server components, at the level of individual facet interfaces.

## Thread Aspect

The baseline Bold Stroke architecture primarily contains four types of threads. There is a single thread that handles external bus specific input and output. There is a set of threads that handle network processing, one for each rate of processing typically resulting from remote Object Request Broker requests. There is another set of threads that handle trigger dispatching and processing, again one for each rate in a rate

monotonic scheduling manner. Finally, there are rare cases of product specific components that have their own threads. These four thread types are representative of most derivative avionics systems, although the balance between the thread types may vary significantly from system to system.

Although a significant amount of research is being applied to more dynamic and adaptive embedded system scheduling, most such systems currently in operational use are statically scheduled. Typically, all threads are created during the initialization and configuration phase of the system. Each thread has a unique static priority. Components have known rates at which they need to produce outputs, and are assigned to threads with related priorities.

Models for the thread subview need to allow the user to:

- Specify the threads in the system.
- Specify the intended rates of execution, which imply deadlines for associated processing.
- Identify the priority assigned to each thread.

## Deployment View

The primary focus of the deployment view is allocation of application components to processors and processes.

## Component Quality of Service Aspect

Components may have various Quality of Service (QoS) attributes that aid in determining how they will be executed, in particular, in which thread the components will execute. Such components are considered independently schedulable. Other components may not have their own QoS attributes. Such components are passive and their execution is purely a function of their invocation by some schedulable component.

A component may have multiple QoS attributes. These attributes are used to schedule the activity of the given component, e.g. determine in which thread a component will run and the order in which the component is activated within a given thread. QoS attributes can include:

- Rates of execution – A range of acceptable rates may be specified or the component may specify that its activity must execute at a specific rate. It is the rate of execution that implicitly defines the component's execution deadline.
- Importance – The relative importance of one component as compared to other

components that have the same rate of execution. This affects the ordering of execution within a thread.

- Execution Time - The amount of processing time that may be used by a component. This is usually the worst case execution time. This may include the execution time of other (passive) components, utilities, and services that this component invokes.

These QoS attributes may remain fixed over the life of a component, or they may change as the state of the system changes. Similarly, a component may exist throughout the life of the system but be designed to be available to respond to triggers during some system states and be dormant and not available during other system states.

Any component quality of service model should allow the user to specify the QoS properties associated with components, including their evolution as the system changes state, and the resulting assignment of components to threads.

## Process Aspect

The first, and relatively simple, step of software allocation is defining the hardware and software processing resources that will be available for component allocation.

Process subview models should allow the user to:

- Define the set of processors and processes available for processing within the system.
- Define the topology of the available resources and the network communication resources available for their interaction.
- Define the allocation of threads to processes.

## Component Allocation Aspect

Allocation of components to hardware resources in distributed systems is influenced by a number of concerns including resource utilization, timing requirements, and unique hardware capabilities.

Component allocation models should allow the user to:

- Identify a group of components that must be allocated to the same processor. This may be done to conserve bandwidth by ensuring that high volume transfers of data are performed locally, or for other reasons. Especially in real-time systems, performance concerns require that components with close communication coupling remain collocated.

- Allocate a component or group of components to a particular processor.
- Automate or aid activities associated with mapping logical system views to the physical deployment of components. For example, in a CORBA based system, automated creation of appropriate CORBA stubs and skeletons when communicating components are placed on different processors.

## Aspect Checking And View Integration

The separation of aspects presents not only a specification challenge, but also a consistency and correctness issue. The first issue is ensuring and/or checking consistency between views and subviews. If there are two views or subviews that are intended to be views of the same system, do they in fact represent the same system? If two views of a system are consistent, and one is modified, does the other view need to be modified to maintain consistency, and if so how? How often should consistency between views be checked? When should consistency be enforced? Many of these questions require development of mappings between the meta-models for each view.

## Model-Based Configuration

Manual creation of configuration data and code associated with all modeling views is a tedious and error prone task in current systems. Although our current approach includes some automation, much of the configuration code is created manually. Aspect models provide an excellent opportunity for automated generation of much of this software. Experiments performed with MoBIES and PCES tools have supported automatic generation of XML-based configuration files for all of the views defined herein. Scripts then translate these XML representations in C++ data tables that are used to configure the system during initialization.

## Conclusion

We are involved in several activities on the MoBIES and PCES programs that have implemented and demonstrated these techniques. Our role on these programs is to define real-world technical challenges to provide a context for research, provide an Open Experimental Platform including a reusable CORBA-based middleware framework and application components for integrated experimentation, collaborate with a wide range of technology developers to define an open tool integration framework that fits our proposed model-driven development process, and define and

execute a set of experiments to evaluate research products. This paper itself describes some of the key aspects that have been isolated, modeled, analyzed, and composed in pursuit of these goals.

---

<sup>ii</sup> Portions of this work were sponsored by the DARPA Information Exploitation Office Model-Based Integration of Embedded Systems program under contract F33615-00-C-1704, and the Program Composition for Embedded Software program under contract F33615-00-C-3048, administered by the Air Force Research Laboratory, Wright-Patterson Air Force Base, USA.

<sup>ii</sup> Winter, Don C., 1996, "Modular, Reusable Flight Software For Production Aircraft", *15<sup>th</sup> AIAA/IEEE Digital Avionics Systems Conference* Proceedings, p. 401-406.

<sup>iii</sup> Sharp, David C., 1998, "Reducing Avionics Software Cost Through Component Based Product Line Development", *Software Technology Conference*.

<sup>iv</sup> Roll, Wendy C., 2003, "Towards Model-Based and CCM-Based Applications for Real-Time Systems", *IEEE International Symposium On Real-Time Distributed Computing*, accepted for publication.