UNIT-VI

A-7E Avionics System: A Case Study in Utilizing Architectural Structures

An object-oriented program's runtime structure often bears little resemblance to its code structure. The code structure is frozen at compile-time; it consists of classes in fixed inheritance relationships. A program's runtime structure consists of rapidly changing networks of communicating objects. In fact, the two structures are largely independent. Trying to [understand] one from the other is like trying to understand the dynamism of living ecosystems from the static taxonomy of plants and animals, and vice versa.

software architecture describes elements of a system and the relations among them. We also emphasized that every system has many kinds of elements and that different architectural structures are useful, even necessary, to present a complete picture of the architecture of a system. Each structure concentrates on one aspect of the architecture.

case study of an architecture designed by engineering and specifying three specific architectural structures: *module decomposition, uses,* and *process*. We will see how these structures complement each other to provide a complete picture of how the system works, and we will see how certain qualities of the system are affected by each one. Table 3.1 summarizes the three structures we will discuss.

Table 3.1. The A-7E's Architecural Structures

Structure	Elements	Relation among Elements	Has InfluenceOver
Module Decomposition	Modules (implementation units)	Is a submodule of; shares a secret with	Ease of change
Uses	Procedures	Requires the correct presence of	Ability to field subsets and develop incrementally
Process	Processes; thread of procedures	Synchronizes with; shares CPU with; excludes	Schedulability; achieving performance goals through parallelism

<u>UNIT-VI</u>

Architecture for the A-7E Avionics System

The architecture for the A-7E avionics system is centered around three architectural structures

- Decomposition, a structure ofmodules
- Uses, a structure ofmodules
- Process, a structure of components andconnectors

We will discuss each in turn.

DECOMPOSITION STRUCTURE

How the A-7E Module Decomposition Structure Achieves Quality Goals

Goal	HowAchieved
Ease of change to: weapons, platform, symbology, input	Information hiding
Understand anticipatedchanges	Formal evaluation procedure to take advantageof experience of domain experts
Assign work teams so that their interactions were minimized	Modules structured as a hierarchy; each work team assigned to a second-level module and all of its descendants

USES STRUCTURE

How the A-7E Uses Structure Achieves Quality Goals

Goal	HowAchieved

UNIT-VI

Incrementally build and test system functions

How the A-7E Uses Structure Achieves Quality Goals

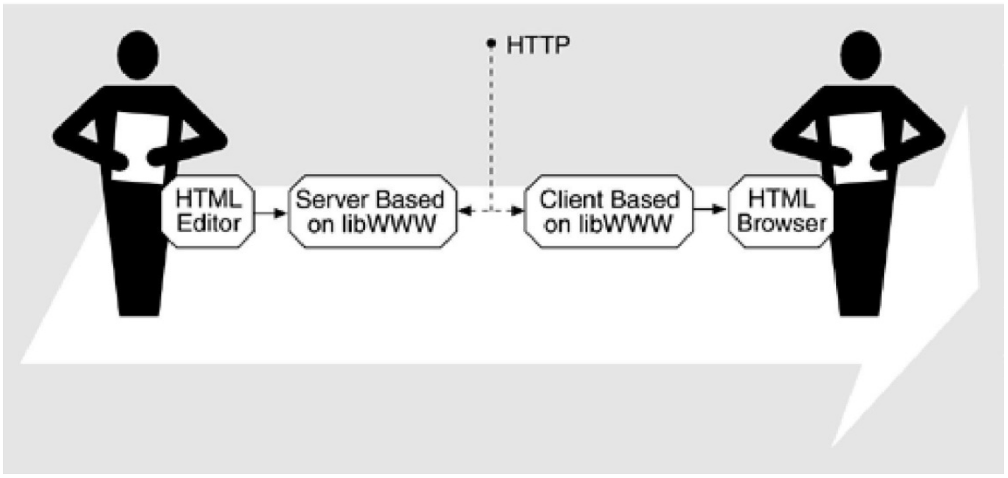
Goal	HowAchieved
Design forplatform change	Restrict number of procedures that use platform directly
Produce usage guidanceof manageable size	Where appropriate, define uses to be a relationship among modules

The World Wide Web'A Case Study in Interoperability

Architectural Solution

The basic architectural approach used for the Web, first at CERN and later at the World Wide Web Consortium (W3C), relied on clients and servers and a library (libWWW) that masks all hardware, operating system, and protocol dependencies. Figure 13.3 shows how the content producers and consumers interact through their respective servers and clients. The producer places content that is described in HTML on a server machine. The server communicates with a client using the HyperText Transfer Protocol (HTTP). The software on both the server and the client is based on libWWW, so the details of the protocol and the dependencies on the platforms are masked from it. One of the elements on the client side is a browser that knows how to display HTML so that the content consumer is presented with an understandable image.

Content producers and consumers interact through clients and servers



consumers interact through clients and servers

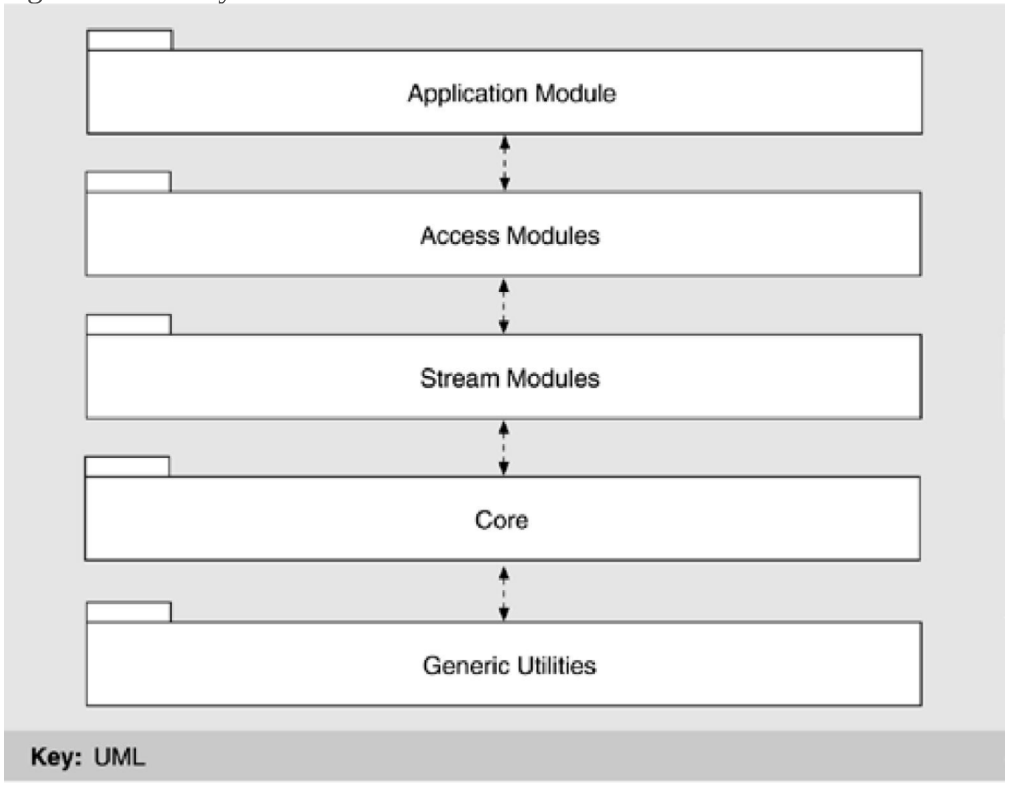
UNIT-VI

MEETING THE ORIGINAL REQUIREMENTS: libWWW

As stated earlier, libWWW is a library of software for creating applications that run on either the client or the server. It provides the generic functionality that is shared by most applications: the ability to connect with remote hosts, the ability to understand streams of HTML data, and so forth.

libWWW is a compact, portable library that can be built on to create Web-based applications such as clients, servers, databases, and Web spiders. It is organized into five layers, as

Figure 13.4. A layered view of libWWW



ACHIEVING INITIAL QUALITY GOALS

describes how the Web achieved its initial quality goals of remote access, interoperability, extensibility, and scalability.

How the WWW Achieved Its Initial Quality Goals

Goal	HowAchieved	TacticsUsed

<u>UNIT-VI</u>

How the WWW Achieved Its Initial Quality Goals

Goal	HowAchieved	TacticsUsed
RemoteAccess	Build Web on topofInternet	Adherenceto definedprotocols
Interoperability	Use libWWW to mask platform details	Abstractcommon services Hide information
Extensibility of Software	Isolate protocol and data type extensions in libWWW; allow for plug-in components (applets and servlets)	Abstract common services Hide information Replace components Configuration files
Extensibility of Data	Make each data item independent except for references it controls	Limit possible options
Scalability	Use client-server architectureand keep references to other data local to referring data location	Introduce concurrency Reduce computational overhead

UNIT-VI

Air Traffic Control: A Case Study in Designing for High Availability

Air traffic control (ATC) is among the most demanding of all software applications. It is *hard real time*, meaning that timing deadlines must be met absolutely; it is *safety critical*, meaning that human lives may be lost if the system does not perform correctly; and it is*highly distributed*, requiring dozens of controllers to work cooperatively to guide aircraft through the airways system. In the United States, whose skies are filled with more commercial, private, and military aircraft than any other part of the world, ATC is an area of intense public scrutiny. Aside from the obvious safety issues, building and maintaining a safe, reliable airways system requires enormous expenditures of public money. ATC is a multibillion-dollar undertaking.

This chapter is a case study of one part of a once-planned, next-generation ATC system for the United States. We will see how its architecture?in particular, a set of carefully chosen views (as in Chapter 2) coupled with the right tactics (as in Chapter 5)?held the key to achieving its demanding and wide-ranging requirements. Although this system was never put into operation because of budgetary constraints, it was implemented and demonstrated

that the system could meet its quality goals.

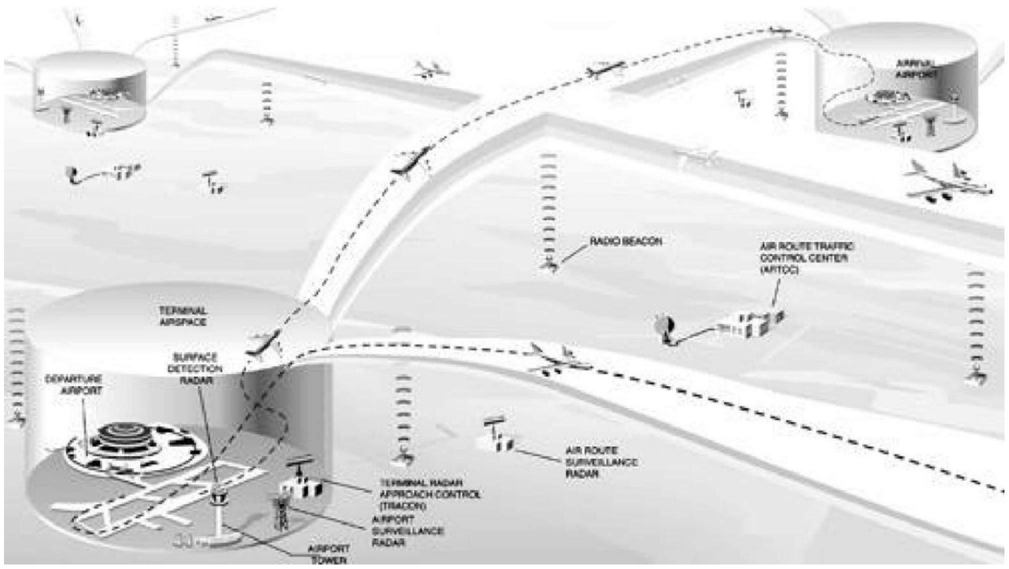
In the United States, air traffic is controlled by the Federal Aviation Administration (FAA), a government agency responsible for aviation safety in general. The FAA is the customer for the system we will describe. As a flight progresses from its departure airport to its arrival airport, it deals with several ATC entities that guide it safely through each portion of the airways (and ground facilities) it is using. *Ground control* coordinates the movement of aircraft on the ground at an airport. Towers control aircraft flying within an airport's*terminal control area*, a cylindrical section of airspace centered at an airport. Finally, *en route centers* divide the skies over the country into 22 large sections of

responsibility.

Consider an airline flight from Key West, Florida, to Washington, D.C.'s Dulles Airport. The crew of the flight will communicate with Key West ground control to taxi from the gate to the end of the runway, Key West tower during takeoff and climb-out, and then Miami Center (the en route center whose airspace covers Key West) once it leaves the Key West terminal control area. From there the flight will be handed off to Jacksonville Center, Atlanta Center, and so forth, until it enters the airspace controlled by Washington Center. From Washington Center, it will be handed off to the Dulles tower, which will guide its approach andlanding.

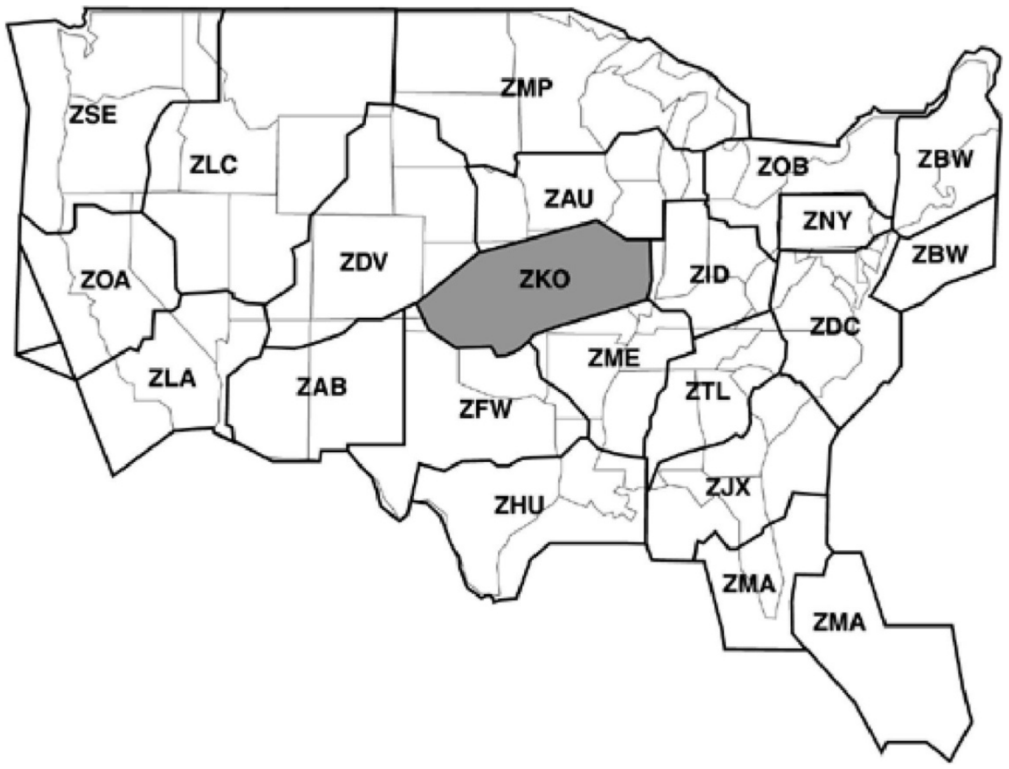
Flying from point A to point B in the U.S. air traffic control system.

Courtesy of Ian Worpole/ *Scientific American* , 1994.


ARRIVAL AIRPORT
RADIO BEACON
AIR ROUTE TRAFFIC CONTROL CENTER (ARTCC)
TERMINAL AIRSPACE
SURFACE DETECTION RADAR
DEPARTURE AIRPORT
AIR ROUTE SURVEILLANCE RADAR
TERMINAL RADAR APPROACH CONTROL (TRACON)
AIRPORT SURVEILLANCE RADAR
AIRPORT TOWER

When it leaves the runway, the flight will communicate with Dulles ground control for its taxi to

Figure 6.2.En route centers in the United States



the gate. This is an oversimplified view of ATC in the United States, but it suffices for our case study. Figure 6.1 shows the hand-off process, and Figure 6.2 shows the 22 enroutecenters.

UNIT-VI

The system we will study is called the Initial Sector Suite System (ISSS), which was intended to be an upgraded hardware and software system for the 22 en route centers in the United States. It was part of a much larger government procurement that would have, in stages, installed similar upgraded systems in the towers and ground control facilities, as well as the transoceanic ATC facilities.

The fact that ISSS was to be **procured as only one of a set of strongly related systems had a profound effect on its architecture. In particular**, there was great incentive to adopt common designs and elements where possible because the ISSS developer also intended to bid on the other systems. After all, these different systems (en route center, tower, ground control) share many elements: interfaces to radio systems, interfaces to flight plan databases, interfaces to each other, interpreting radar data, requirements for reliability and

performance, and so on. Thus, the ISSS design was influenced broadly by the requirements for all of the upgraded systems, not just the ISSS-specific ones. The complete set of upgraded systems was to be called the Advanced Automation System (AAS).

Ultimately, the AAS program was canceled in favor of a less ambitious, less costly, more staged upgrade plan. Nevertheless, ISSS is still an illuminating case study because, when the program was canceled, the design and most of the code were actually already completed. Furthermore, the architecture of the system (as well as most other aspects) was studied by an independent audit team and found to be well suited to its requirements. Finally, the system that was deployed instead of ISSS borrowed heavily from the ISSS architecture. For these reasons, we will present the ISSS architecture as an actual solution to an extremely difficult problem.

How the ATC System Achieves Its Quality Goals

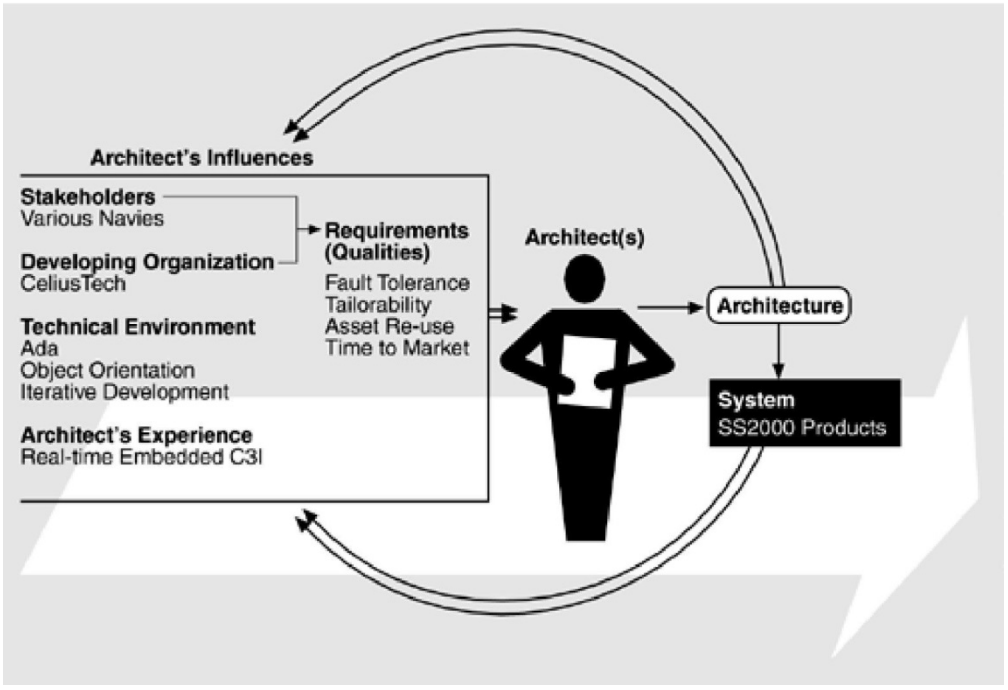
Goal	HowAchieved	Tactic(s) Used
High Availability	Hardware redundancy (both processor and network); software redundancy (layered fault detection and recovery)	State resynchronization; shadowing; active redundancy; removal from service; limit exposure; ping/echo; heartbeat; exception; spare
High Performance	Distributed multiprocessors; front-end schedulability analysis, and network modeling	Introduce concurrency
Openness	Interface wrappingand layering	Abstract common services; maintain interface stability
Modifiability	Templates and table-driven adaptation data; careful assignment of module responsbilities; strict use of specified interfaces	Abstract common services; semantic coherence; maintain interface stability; anticipate expected changes; generalize the module; component replacement; adherence to defined procotols; configuration files
Ability to Field Subsets	Appropriate separation of concerns	Abstract common services
Interoperability	Client-server divisionof	functionality and message-

CelsiusTech: A Case Study in Product Line Development

This chapter relates the experience of CelsiusTech AB, a Swedish naval defense contractor that successfully adopted a product line approach to building complex software-intensive systems. Called Ship System 2000 (SS2000), their product line consists of shipboard command-and-control systems for Scandinavian, Middle Eastern, and South Pacific navies.

This case study illustrates the entire Architecture Business Cycle (ABC), but especially shows how a product line architecture led CelsiusTech to new business opportunities. Figure 15.1 shows the roles of the ABC stakeholders in the CelsiusTech experience.

Figure 15.1. The ABC as applied to CelsiusTech



SS2000 Requirements and How the Architecture Achieved Them

Requirement	HowAchieved	RelatedTactic(s)

UNIT-VI

SS2000 Requirements and How the Architecture Achieved Them

Requirement	HowAchieved	RelatedTactic(s)
Performance	Strict network traffic protocols; software is written as a set of processes to maximize concurrency and written to be location independent, allowing for relocation to tune performance; COOB is by-passed for high-data-volume transactions; otherwise, data sent only when altered and distributed so response times are short	Introduce concurrency Reduce demand Multiple copies Increase resources
Reliability, Availability, and Safety	Redundant LAN; fault-tolerant software; standard Ada exception protocols; software written to be location independent and hence can be migrated in case of failure; strict ownership of data prevents multi-writer race conditions	Exceptions Active redundancy State resynchronization Transactions
Modifiability (including ability to produce new members of the SS2000 family)	Strict use of message-based communication provides interface isolated from implementation details; software written to be location independent; layering provides portability across platforms, network topologies, IPC protocols, etc.; data producers and consumers unaware of each other because of COOB; heavy use of Ada;	

UNIT-VI

SS2000 Requirements and How the Architecture Achieved Them

Requirement	HowAchieved	RelatedTactic(s)
		Configuration files Component replacement Adherence to defined protocols

UNIT-VI

Testability	Interfaces using strongly typed messages push a whole class of errors to compile time; strict data ownership, semantic coherence of elements, and strong interface definitions simplify discovery of responsibility