Chapter **8**

Physical Architecture Development

8.1 INTRODUCTION

The physical architecture of a system is a hierarchical description of the resources that comprise the system. This hierarchy begins with the system and the system's top-level components and progresses down to the configuration items (CIs) that comprise each intermediate component. The CIs can be hardware or software elements or combinations of hardware and software, people, facilities, procedures, and documents (e.g., user's manuals).

Section 8.2 introduces the distinction between a generic and instantiated physical architecture. The generic physical architecture defines the hierarchy in general terms, for example, two processors with associated software, a person, and a building. The instantiated physical architecture lays out the specifics of the processors, software, person, and building in enough detail to permit performance modeling of the system related to the requirements being addressed. The intent of systems engineers should not be to design these components but rather to state representative instantiations for the generic components that are sufficient to model the performance of the system and ensure that the requirements decomposition process makes sense.

Section 8.3 defines a method for developing alternatives for the generic and instantiated physical architectures of the system. The development process proposed here emphasizes multiple alternatives, especially for the instantiated physical architecture, based on the supposition that the design process is quite difficult for even moderate extensions of existing systems. The following quote

The Engineering Design of Systems: Models and Methods, Second Edition. By Dennis M. Buede

by Guindon [1990, p. 308] expresses the importance of this approach:

> System design often involves novelty. Even though the designer may be thoroughly familiar with the design process itself, there may not be any precedent in the literature for the system to be designed. — It may be a new technology. More frequently, the system may simply involve some novelty in an otherwise well-understood problem. The novelty may range from a novel combination of requirements for a familiar type of system in a familiar problem domain. As a consequence, there is often no predetermined solution path from the requirements to the finished artifact [Newell, 1969; Nii, 1986; Reitman, 1965; Rittel, 1972; Simon, 1973]. Thus system design frequently requires the creation of new solutions interleaved with the application of known solutions.

Section 8.4 introduces some creativity techniques to aid in the development of the alternate physical architectures. The morphological box is the primary technique employed and illustrated in this chapter. The morphological box dates back to the 1940s and breaks a system into segments as defined by the generic physical architecture; it then provides for the listing of alternate instantiated physical components for each segment. Other techniques that have been proposed and utilized are classified as either brainstorming or brainwriting and are also discussed. See West [2007]. Selecting one or more instantiated components from each component produces an alternative for an instantiated physical architecture for the system.

Engineers commonly resort to describing the system's architecture in a non-mathematics-based graphical format. Block diagrams, the commonly used and non-standardized graphical format, are presented in Section 8.5 to represent the physical coupling of the system's components. A block diagram provides a box or block for each component. The links between the blocks represent the major flows of energy or information between the components represented by the blocks.

Section 8.6 addresses major issues and associated concepts in the development of a physical architecture. The concepts of centralized and decentralized, distributed, and client–server architectures are discussed and illustrated. Also redundancies in hardware, software, information, and time are discussed as ways to achieve fault tolerance via the physical architecture.

The exit criterion for the development of the physical architecture is the provision of a single physical architecture that is satisfactory in terms of detail, quantity, and quality for development of the allocated architecture. This satisfaction of detail, quantity, and quality is typically preceded by the creation of several alternate physical architectures for consideration during the development and refinement of the allocated architecture.

8.2 GENERIC VERSUS INSTANTIATED PHYSICAL ARCHITECTURES

The *physical architecture* provides resources for every function identified in the functional architecture. Since every phase of the life cycle is addressed in the requirements and is being addressed in the functional architectures, there must be a physical architecture for each system associated with the system's life cycle. Recall the sample physical architecture from Chapter 1 (repeated here as Figure 8.1). Note that this physical architecture includes the vehicle, the support resources for the vehicle during the operational and maintenance phases, and the training resources, which may be training for the operational phase or the training phase. Also, note that even at the third level of the physical architecture, the components are combinations of hardware, software, and other devices.

Military standard MIL-STD-881B [1993] contains a Work Breakdown Structure (WBS) for Defense Material Items. The WBS is often very similar to the physical architecture because the work is organized along the lines of the resources that require development or procurement. For an aircraft system there are 10 elements that partition the system, as shown in the first column of Table 8.1. These elements span six of the seven life-cycle phases (shown in the second column) defined in Chapter 1. The only phase that is absent from this list is retirement, the commonly forgotten phase.

In the same military standard, 17 resource categories, shown in Table 8.2, are defined as a partition of the generic air vehicle. These lists or partitions of the resources for the physical architecture are most useful as memory joggers. For some aircraft, some of these elements are not relevant; for example, airlift aircraft do not need armament or antisubmarine warfare. More importantly, as technology advances some of these elements are outdated. With the advent and advance of distributed computing, the central computer element is not relevant or misleading. In addition, at this level of the physical architecture it is often too early to separate hardware and software.

Common resource categories for an aircraft have been described in Figure 8.1 and Tables 8.1 and 8.2. The resource categories for the elevator's physical architecture from the case study, which can be downloaded from

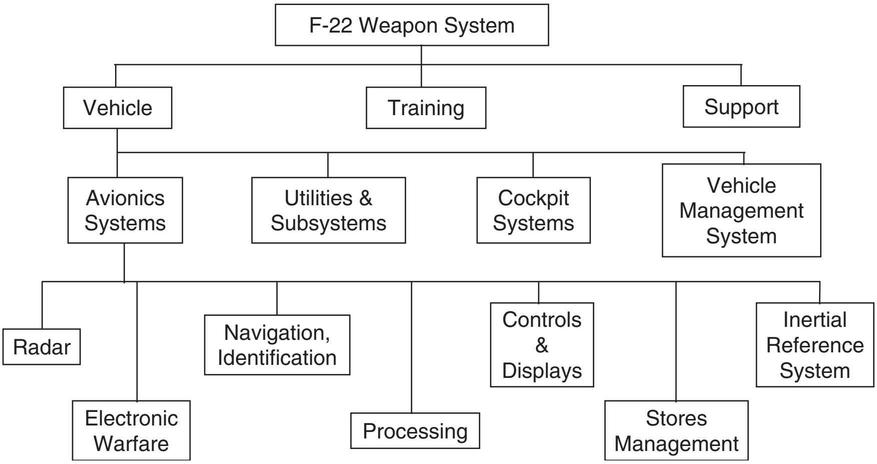


FIGURE 8.1 Sample physical architecture (F-22 Type A Spec) (from Reed [1993]).

TABLE 8.1 WBS Elements and Related Life Cycle Phases

WBS Elements	Life Cycle Phase
Air vehicle	Operational
Systems engineering/Program management	Development
System test and evaluation	Development
Training	Training
Data	Manufacturing and Refinement
Peculiar support equipment	Operational
Common support equipment	Operational
Operational/site activation	Deployment
Industrial facilities	Manufacturing
Initial spares and repair parts	Operational

http://www.theengineeringdesignofsystems.com, are shown in Figure 8.2. All of these resource categories are examples of a generic physical architecture. A *generic physical architecture* is a description of the partitioned elements of the physical architecture without any specification of the performance characteristics of the physical resources that comprise each element (e.g., central processing unit). An *instantiated physical architecture* is a generic physical architecture to which complete definitions of the performance characteristics of the resources have been added. An instantiated physical architecture for the elevator system would be specific about the call announcement component (e.g., liquid crystal lights), destination control (e.g., push buttons), and the like.

One element that is left out of most physical architectures is the set of procedures that are developed for the users of the system to follow. These *procedures* are explicit operating, maintenance, or support instructions provided in the font of a user's or operator's manual. These manuals usually accompany the system when the system is delivered. These procedures are the focus of attention during the training that is delivered to the users, maintainers,

TABLE 8.2 Resource Categories for a Generic Air Vehicle

- Airframe
- Propulsion
- Air vehicle application software
- Air vehicle system software
- Communications/Identification
- Navigation/Guidance
- Central computer
- Fire control
- Data display and controls
- Survivability
- Reconnaissance
- Automatic flight control
- Central integrated checkout
- Antisubmarine warfare
- Armament
- Weapons delivery
- Auxiliary equipment

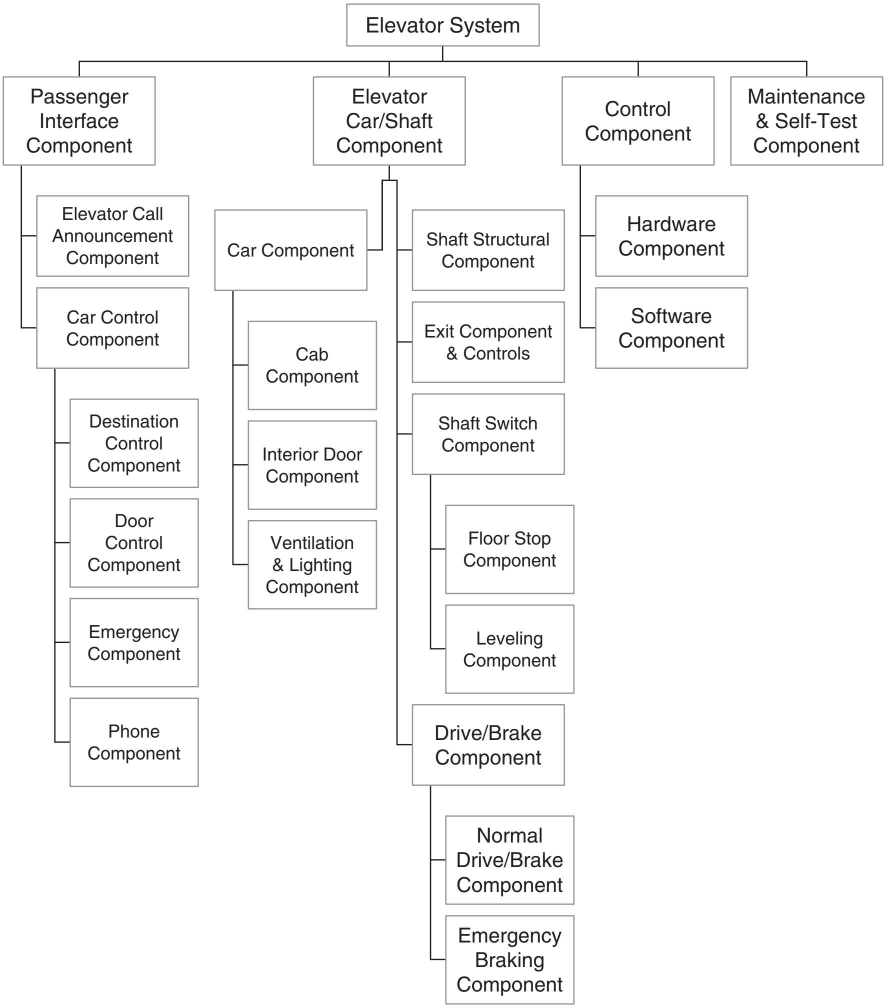

Elevator System
Passenger Interface Component
Elevator Car/Shaft Component
Control Component
Maintenance & Self-Test Component
Elevator Call Announcement Component
Car Control Component
Destination Control Component
Door Control Component
Emergency Component
Phone Component
Car Component
Cab Component
Interior Door Component
Ventilation & Lighting Component
Shaft Structural Component
Exit Component & Controls
Shaft Switch Component
Floor Stop Component
Leveling Component
Drive/Brake Component
Normal Drive/Brake Component
Emergency Braking Component
Hardware Component
Software Component

FIGURE 8.2 Generic physical architecture from the elevator case study.

or supporters of the system. Systems engineers should not forget or ignore this element of the system's physical architecture, as was done with the initial air bag system that was described as a case study in Chapter 6. After the serious, and often deadly, effects on children and small adults were noticed, a series of procedures for the placement (or lack thereof) of children and small adults in the front seat were released. Common practice in the development of a system is to accommodate problem issues identified during qualification of the system (see Chapter 10) by amending and expanding the procedures defining how the system will be used. *Procedures such as these represent the way in which the system's functionality moves from the system under development to the users.*

8.3 OVERVIEW OF PHYSICAL ARCHITECTURE DEVELOPMENT

The definition of the physical architecture, as described here, is done one level of the tree at a time. Our approach here is a top-down process. There are many systems engineers that have successfully used a bottom-up design process for the physical part of the system (just as we described the bottom-up approach in the previous chapter for the functional architecture). Experience and creativity are critical for this part of the engineering process. While experience is a must; do not underestimate the importance of creativity.

There are many possible decompositions of the process "Design System Physical Architecture." The one chosen here (Figure 8.3, taken from Appendix B) emphasizes the concepts of generic and instantiated physical architectures. A second justification of this decomposition is the belief that the allocated architecture development is predicated on having a variety of interesting physical architectures to match with the functional architecture. Therefore, the primary product of this function for designing the physical architecture is a reasonable number of interesting physical architectures that can be combined with the functional architecture and evaluated to determine their effectiveness in meeting the objectives established in the requirements.

The structure of the generic physical architecture is first selected while working in parallel with the development of the functional architecture. As discussed in Chapter 7 and elaborated on in Chapter 9, there are great advantages in defining the internal interfaces of the system to have the functional and physical architectures match; that is, enable a one-to-one and onto allocation of functions to components. See Figure 8.4 to review the distinctions between a relation and a function, and the additional restrictions for a function that is one-to-one and onto. While there are many advantages to a one-to-one and onto mapping of functions and components, this may not always be possible and should not be forced.

First, a generic physical architecture must be developed. The generic physical architecture provides common designators for physical resources in a hierarchical decomposition that partitions the system into greater and greater detail. Although this generic physical architecture has no substance in the sense of specific physical items, this structure is still very important. Some instantiated physical architectures can be eliminated from consideration just on the basis of the division of the system into components. Therefore serious thought and creativity should be devoted to this initial task.

The second function in the decomposition addresses the creation of a morphological box to assist in generating a set of creative instantiated architectures to analyze during the development of the allocated architecture. A *morphological box* is a matrix in which the columns (or rows) represent the components in the generic physical architecture. The boxes in a given column (or row) then represent alternate choices for fulfilling that generic component. Each option should have well-defined performance (and cost) characteristics. Section 1 describes the morphological box in more detail and provides several examples.

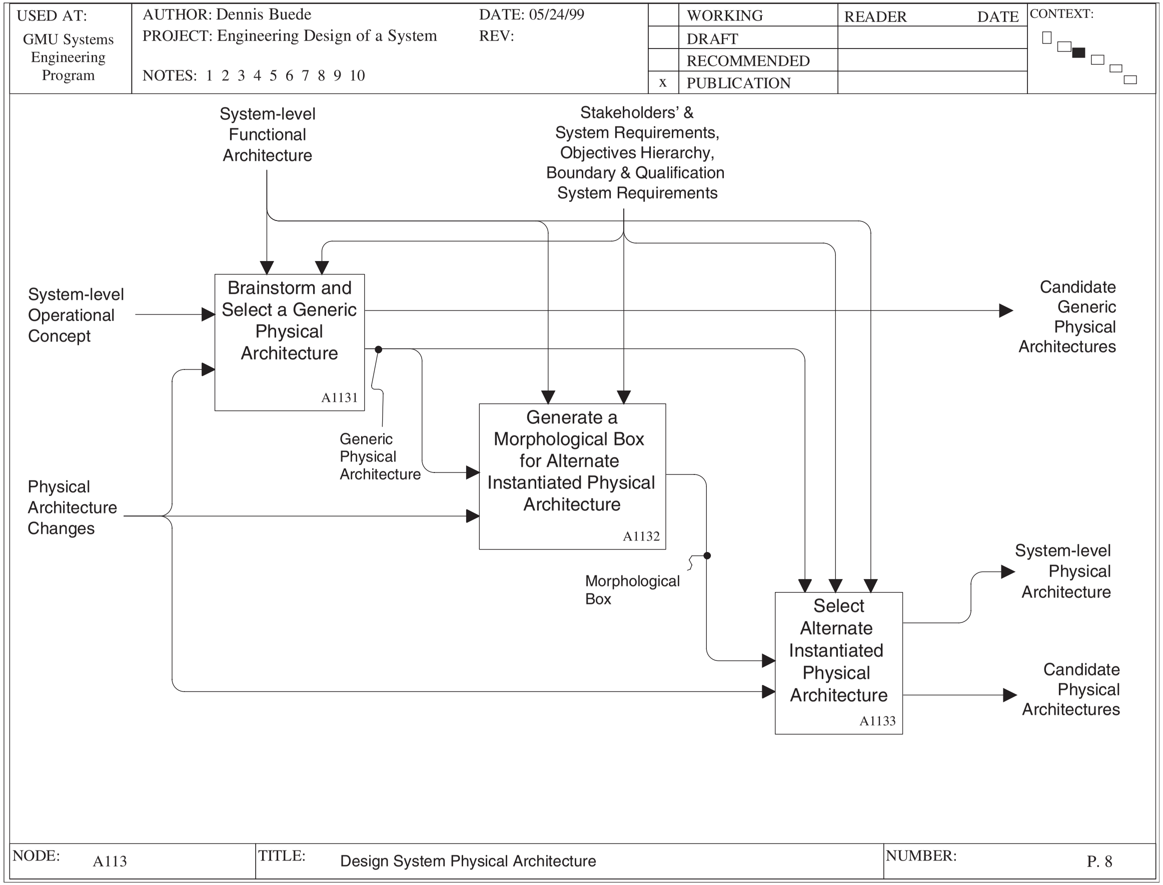
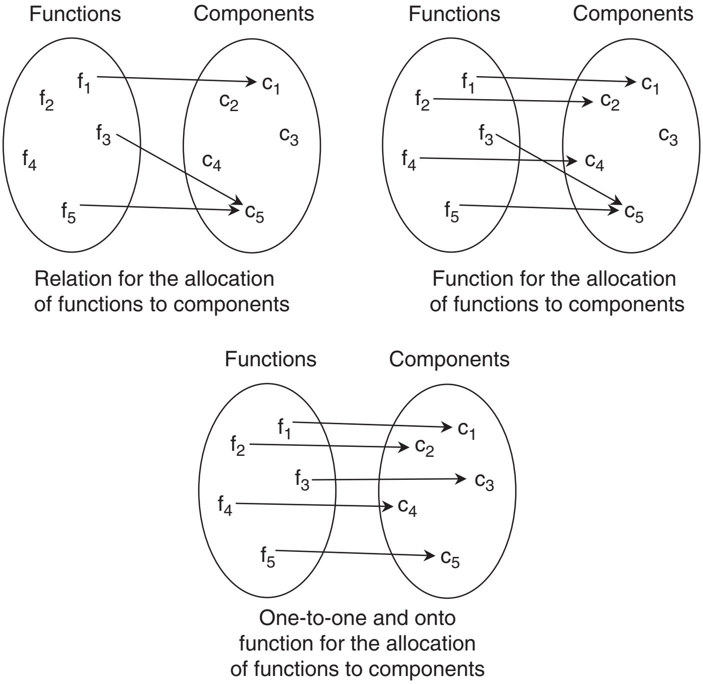

USED AT:
GMU Systems Engineering Program
AUTHOR: Dennis Buede
PROJECT: Engineering Design of a System
NOTES: 1 2 3 4 5 6 7 8 9 10
DATE: 05/24/99
REV:
WORKING
DRAFT
RECOMMENDED
x PUBLICATION
READER
DATE
CONTEXT:
System-level Functional Architecture
Stakeholders' & System Requirements, Objectives Hierarchy, Boundary & Qualification System Requirements
System-level Operational Concept
Physical Architecture Changes
Brainstorm and Select a Generic Physical Architecture
A1131
Generic Physical Architecture
Generate a Morphological Box for Alternate Instantiated Physical Architecture
A1132
Morphological Box
Select Alternate Instantiated Physical Architecture
A1133
Candidate Generic Physical Architectures
System-level Physical Architecture
Candidate Physical Architectures
NODE: A113
TITLE: Design System Physical Architecture
NUMBER: P. 8

FIGURE 8.3 Development process for the physical architecture.



Relation for the allocation of functions to components

Function for the allocation of functions to components

One-to-one and onto function for the allocation of functions to components

FIGURE 8.4 Need for a one-to-one and onto functional allocation of functions to components.

The third function in this decomposition uses the morphological box to aid in the selection of as many alternate instantiated physical architectures as are needed to feed the process of selecting an allocated architecture. An alternative instantiated physical architecture would be the selection of an option from each of the generic components in the morphological box. Examples of the morphological box are provided in the following sections.

The functional decomposition shown in Figure 8.3 suggests that the three functions are performed in a serial fashion, which is true with the following caveat: The changes to the physical architecture that are sent from the development of the allocated architecture trigger the repetition of these three functions. Each repetition could cause changes to the generic physical architecture, modifications to the morphological box due to the changed generic architecture or other changes dictated by the allocated architecture, and a reselection of alternate instantiated physical architectures.

8.4 CREATIVITY TECHNIQUES

Initially creating more choices than are useful to consider in a detailed analysis process is wise. This generation of excess alternatives means there is a greater chance that the best choices are being considered in the final analysis. There are many possible creativity enhancing techniques that have been used by engineers to develop new and interesting solutions to old and new problems. This section begins by focusing on one technique, the morphological box, that has proven useful a number of times. Then a larger review of techniques is provided.

8.4.1 Morphological Box

Originally proposed by Zwicky [1969] during World War II and then expanded by Allen [1962], morphological analysis (more commonly known in some disciplines as morphological box) divides a problem into segments and posits several solutions for each segment. In the two-dimensional version, a table is created with columns (or sometimes rows) pertaining to the generic components of the physical architecture. Then the elements of each column are filled with competing specific instantiations of each component. The instantiations in a given column need not fit together; in fact, each column corresponds to a section of a cafeteria (e.g., salads, vegetables, meat, deserts). A meal would then consist of a selection from each section of the cafeteria. A system's instantiated physical architecture, analogously, is a selection of one box from each column (generic component) of the morphological box. As part of the morphological analysis, each instantiation (one from each column) will be based upon a subset of the system's objectives. For example, one subset of objectives might be low cost; another, high-speed performance; and a third, high usability. Each of these instantiations is, in fact, a theme for the design of the system.

Table 8.3 presents a morphological box (generic components and choices) for a hammer. This morphological box contains five generic components of a hammer: the length of the handle, the material that the handle is made of, the size and surface of the head of the hammer used for striking, the weight or density of the hammer head, and the angle associated with the head of the hammer used for removing nails. Any hammer is one cell from each of the five columns. For example, one hammer design is obtained by taking the top cell of each column: 8-inch handle made of Fiberglass with a rubber grip using a 1 inch diameter flat steel head that weighs 12 ounces and has a steel claw that is nearly perpendicular to the handle. There are $2 \times 5 \times 4 \times 4 \times 2 = 320$ different possible hammers defined in this table, assuming none of the combinations are infeasible. Yet when you go to the hardware store, there may be only a dozen choices. For real systems there are usually millions of possible combinations. Yet many design teams only consider one or two in any detail, making it very likely that they are missing several creative, high-quality designs. The big advantage of the morphological box is that it forces the design team to recognize that there are many possible solutions to the design problem. The conversation about what design alternative best satisfies the requirements follows naturally.

While the morphological box is a simple concept, there are a number of subtle issues that need to be addressed. First and obviously, there should be at

TABLE 8.3 Morphological Box for a Hammer

Handle Size	Handle Material	Striking Element	Weight of Hammer Head	Nail Removal Element
8 inches	Fiberglass with rubber grip	1 inch diameter flat steel	12 oz.	Steel claw at nearly a straight angle
22 inches	Graphite with rubber grip	1 inch diameter grooved steel	16 oz.	Steel claw at a 60 degree angle with handle
	Steel with rubber grip	1.25 inch diameter flat steel	20 oz.	
	Steel I-beam encased in plastic with rubber grip	1.25 inch diameter grooved steel	24 oz.	
	Wood			

least one column in the morphological box for each generic component in the physical architecture. There are certainly situations in which one of the generic components may have two or more columns associated with the generic component; these would be the decomposed generic components of the higher level component.

Second, there is no requirement that each generic component have the same number of options. Clearly, there is value to having at least two choices for any generic component; otherwise that particular generic component has been fixed. Using some of the brainstorming or brainwriting techniques to be discussed in Section 8.4.2 is common to develop additional alternatives (boxes) for each generic component (column of the morphological box). There is great advantage to generating a creative set of choices for any generic component, even if some of the choices are never selected in the final set of alternate instantiated physical architectures.

In addition, there are situations in which it is wise to permit more than one choice from a generic component to be selected for a single instantiated physical architecture. This possibility of selecting several choices in a single generic component for a single instantiated physical architecture usually does not make sense for a central component in the architecture. However, there are often generic components associated with the ‘‘bells and whistles’’ of the system. An example would be the list of peripherals that can be added to a computer or an automobile. There is some efficiency to group all of these under one generic component for the system rather than have a generic component for each of the possible peripherals.

Figure 8.5 provides another example of a morphological box; this example describes alternate designs for an automobile navigation support system. A

Direction Support	Localization	Processor	User I/O	Other System Interfaces
Map & Database	None ◆	None ◆	Regular Cell Phone ✚	None ⬢ ✚ ● ■
Map, Database, Routing Algorithm ⬢ ✚ ● ■	Direction Sensor	Vehicle's Processor	Special Cell Phone ▲ ◆	Horn ▲
Staffed Control Center ▲ ◆	Electro Gyros ✚	32-bit Processor ⬢ ✚ ● ▲	4" LCD ●	Lights ▲
Automated Control Center	GPS Transponder ⬢ ● ◆ ■	Portable PC (486+) ■	6" LCD ✚	Car Door Locks ▲
	Full GPS Support ▲		6" LCD & Touch Screen ⬢	Emergency Signal ✚ ▲ ◆
			Button & Key Panel ⬢ ●	Air Bag ▲
			Joy Stick ⬢	
			Control Knob ✚	
			Voice Output ✚ ■	

⬢	Acura Navigation System	▲	Cadillac's OnStar
✚	BMW Navigation System	◆	Lincoln's RESCU
●	Oldsmobile Guidestar	■	RETKI

FIGURE 8.5 Morphological box for automobile navigation support system.

number of automakers are providing such navigation support systems as peripherals (or extras) now. In addition, a number of peripheral companies are providing such navigation support systems that can be added to any automobile. In general, these navigation support systems provide the driver and passengers with information about where they are on the highway and how to get where they want to go. However, there are extras that can be provided as shown in the last column, "Other System Interfaces." These extras include the ability to have the car doors unlocked when the owner has locked him/herself out, notify the police or emergency service if the air bag deploys, and activate the lights and horn externally if the driver has lost the car in a parking lot. Selecting more than one option in the second to last column is also possible; this column represents the generic component associated with the user interface for the navigation support system. The selection of multiple boxes is also common for user interface generic components.

There is one major caution that must be provided in the development of a morphological box. The system concept has to be narrowed down to some degree before it is possible to define a single morphological box. For example, if the system is a substantial computer system, a morphological box cannot be defined before an architecture for the computer system has been selected. For example, suppose the alternate computer system architectures were a client–server, a mainframe, or a distributed processing architecture connected via several local area networks (LANs). The generic components that are applicable to a client–server architecture may not be consistent with those generic components for a mainframe system or a distributed network. Therefore the design process should narrow the computer system architecture down to a client–server or mainframe before developing a morphological box.

Once a reasonable number of possible choices for each component of the physical architecture have been identified, identifying infeasible combinations may be wise. Friend and Hickling [1987] have defined a graphical representation to highlight pairwise infeasible choices across two generic components. Each generic component is shown as a circular node in a graph. The specific choices for a generic component are shown as pie-shaped wedges in the relevant generic component's node. An infeasible combination of choices from two distinct generic components is shown as a line between those options.

Pairwise examples of infeasible combinations are shown in Figure 8.6 for the morphological box of the hammer shown in Table 8.3. In this hypothetical example the line segment from angled nail removal feature to 22-inch handle denotes an infeasible combination; an angled nail removal claw cannot be placed on a 22-inch handle because too much stress would be focused at the intersection of the handle and hammer's head. The second line segment shown between the 22-inch handle and a wood handle eliminates the ability of the user

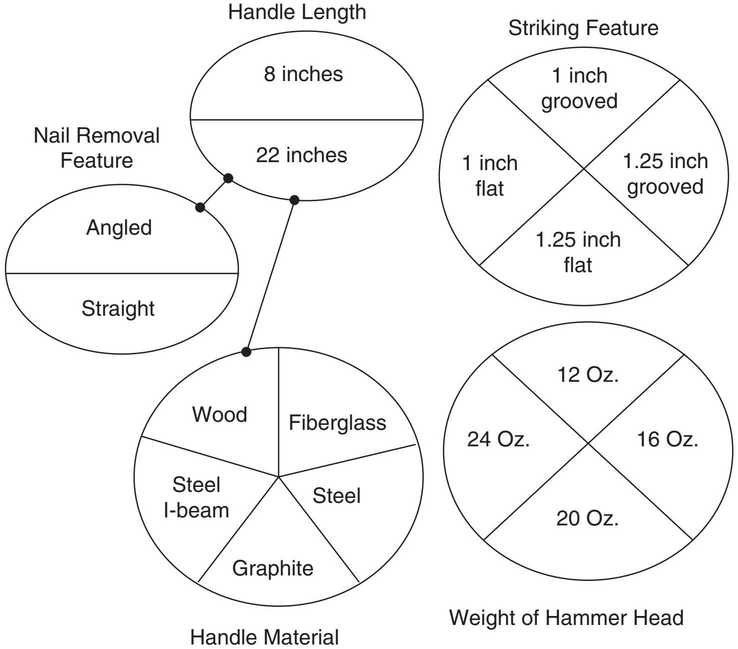


FIGURE 8.6 Pairwise infeasible combinations.

to apply too much force for the wood handle to absorb. These two line segments reduce the total number of choices from 320 to 224; the 8-inch handle still retains 160 possible combinations, but the 22-inch handle only has 64 possible combinations — any of the four striking surfaces with any of the four weights with the one nail removal generic component with four of the five possible handle material generic components.

8.4.2 Option Creation Techniques

VanGundy [1988] is an excellent source of brainstorming techniques and has produced a typology of techniques involving brainwriting or brainstorming; see Table 8.4. *Brainstorming* is the generation of ideas via verbal interaction. *Brainwriting* is a silent, writing process. VanGundy claims:

> Brainstorming, for example, is most useful when there is only a small group of individuals, time is plentiful, status differences among group members are minimal, and a need exists to verbally discuss ideas with others. Brainwriting, on the other hand, is most useful for very large groups, when there is little time available, status differences need to be equalized, and there is no need for verbal interaction. In addition, brainwriting often will produce more ideas than brainstorming, although the uniqueness and quality of these ideas might or might not be superior to those produced by brainstorming. [VanGundy, 1988, p. 75].

A common characteristic, called deferred judgment, of brainstorming and brainwriting exercises is that the individual or group operates in an evaluation-free period where criticism and discussion in general is prohibited. The logic for this freethinking period is that even the most preposterous idea may stimulate the generation of a really superior idea. A second principle is that the more ideas generated the better the chance of finding a high-quality solution. Several techniques discussed below are analogy, people involved, attribute listing, collective notebook, brainwriting game, and brainwriting pool.

Analogies are often used in systems engineering because building upon our experiences with previous systems has a great deal of creative power. An example of an analogy would be to use the 17 elements of the generic aircraft in Table 8.2 to develop a physical architecture of an automobile, an air traffic control system, or an elevator system. Using the physical architecture from a system recently developed as an analogy for a new generation product is another example of analogic reasoning. The use of analogies for generating ideas is by far the most common, efficient, and highly recommended; however, left unchecked analogic reasoning can produce the most disastrous results.

Examining the system's physical architecture in light of the stakeholders (*people involved*) affected by the use and maintenance of the system can be useful in defining the physical architecture for the operational phase. Remember though that the entire life cycle of the system must be addressed, so there

TABLE 8.4 VanGundy's Typology of Brainwriting and Brainstorming

Brainwriting and Brainstorming Categories	Examples
Brainwriting I—an individual works alone to create a list of ideas.	Analogy, Attribute Listing, People Involved
Brainwriting II—a group of individuals separated in space generates ideas separately and the ideas are collected but not shared	Collective Notebook
Brainwriting III—a group of individuals separated in space generates ideas separately, the ideas are shared and additional ideas are generated	Delphi Method
Brainwriting IV—a group of individuals working in the same room generates ideas separately and the ideas are collected but not shared and no discussion takes place	Nominal Group Technique
Brainwriting V—a group of individuals working in the same room generates ideas separately; all of the ideas are shared but none are discussed; additional ideas are generated	Brainwriting Pool
Brainstorming I—a group of individuals generates ideas via verbal discussion, no defined procedure is used	Unstructured Group Discussion
Brainstorming II—a group of individuals generates ideas via verbal discussion within the bounds of pre-defined procedures	Classical Brainstorming
Brainwriting/Brainstorming I—a group of individuals generates ideas via predefined written and verbal procedures	Brainwriting Game

will be physical architectures for the manufacturing, deployment, and training phases as well.

Attribute listing dates back to the 1930s and is based on the concept that physical architectures can all be traced to modifications of previous architectures. Once the requirements and objectives of the system have been developed and a generic physical architecture has been created, the individual defines a feasible (or nearly feasible) instantiation of the generic physical architecture. Then without detailed evaluation, she systematically modifies the characteristics of the instantiated physical architecture with key objectives of the system in mind. For example, VanGundy provides the following example for a hammer:

> To develop a better hammer, for example, the following parts could be listed: (1) straight, wooden, varnished handle; (2) metal head with round striking surface on one end and a claw on the other; and (3) metal wedge in the top of the handle to secure the head to the handle. Of these parts, the basic attributes of handle shape/composition and the metal wedge could be selected for possible modification. The handle could be constructed of fiberglass, wrapped with a shock-absorbing

> material, and shaped to better fit the human hand; the metal wedge could be modified by replacing it with a synthetic, pressure-treated bonding. [VanGundy, 1988, p. 88]

Morphological analysis (sometimes called matrix analysis) results in a morphological box, which is a systematic extension of attribute listing. This topic was discussed in detail with examples above.

Haefele [1962] of the Proctor and Gamble Company developed the Collective Notebook. Each participant in this group-oriented technique keeps a notebook of ideas over a relatively long time period to solve a specified problem; Haefele suggested one month. Each participant is to add one idea each day. At the end of idea collection period, each participant reviews her own ideas and selects the best one; ideas needing more research or other good ideas that may relate to other problems are annotated. A coordinator, who collects this summary information and the notebooks, creates a detailed synopsis of the ideas generated that can then be reviewed by the participants.

The *brainwriting game* uses competition among the participants to create the most improbable solution in hopes that this competition will generate the best solution. First, the design problem is presented to the group. Each participant buys a specified number of blank, numbered cards. The participant places her initials on her cards and then writes an idea that she hopes will win the prize for the most improbable solution. All of the cards are then displayed to the entire group. Participants then individually write more practical solutions based upon concepts taken from the cards detailing improbable solutions. After the practical solutions are collected, the group votes on the winner of the most improbable solution. Finally, subgroups are formed that then work on similar, practical solutions. Finally the group selects its best idea(s).

The *brainwriting pool* involves a group of five to eight people. The group leader presents the design problem to the group and each individual begins writing solutions on a piece of paper. As soon as each individual gets four solutions documented, he places his paper in the middle of the table and selects a paper from someone else. He then reviews the ideas on that paper and adds new ideas triggered from reading the list. After placing another few ideas on that paper, he exchanges it for another paper in the middle of the table. This continues for 20 to 30 minutes. The group then reviews the ideas.

In addition to the techniques summarized by VanGundy [1988], Altshuller [Arciszewsti, 1985, Terninks et al., 1996] began the development of a theory of inventive problem solving (TRIZ) for product development in Russia in 1946. TRIZ is the result of the analysis of approximately 1.5 million patents from across the world. The problem-solving methods employed in TRIZ include Altshuller's inventive principles, table for engineering contradiction elimination, standard techniques to eliminate conflicts, standard solutions to inventive problems, and algorithm for inventive problem solving. This material is still largely proprietary and is marketed by a number of consultants and seminar leaders.

An important creativity concept with which to finish draws upon the notions of value-focused thinking [Keeney, 1992], introduced in Chapter 6. This approach is similar to the attribute listing method discussed above. The individual selects one or more important key performance requirements and defines an instantiated physical architecture or choices within a single generic component. Then another single performance requirement or set of performance requirements is selected and used to generate an instantiated architecture or set of choices for a single generic component. After continuing this process for a productive period of time, the results are critiqued and adapted to feasible solutions.

8.5 GRAPHIC REPRESENTATIONS OF THE PHYSICAL ARCHITECTURE

There are many graphical representations of a physical architecture with little standardization. The most common graphical format is called a *block diagram*. Figure 8.7 illustrates a block diagram for the control system of an aircraft. Each box inside the dotted line defining the control system represents a physical component of the control system. The lines between the boxes indicate the flow of electromechanical energy between the boxes. The boxes outside the dotted line represent other components of the aircraft system. This block diagram shows a decentralized controller structure in which there is a central controller and an actuator controller for each device actuator. Note the feedback loops inside the control component, as well as the feedback loop involving most of the elements of the control component and the actuator devices that are part of the aircraft but outside the aircraft control system.

There was no accepted convention for block diagrams prior to SysML, which was introduced in Chapter 3. SysML contains two types of block diagrams: block definition diagrams and internal block diagrams. The block definition diagram (see Figure 3.14) shows the hierarchical decomposition shown in Figure 8.1. The internal block diagram (see Fig. 3.16) presents the information shown in the generic block diagram of Figure 8.7.

8.6 ISSUES IN PHYSICAL ARCHITECTURE DEVELOPMENT

The major issues in designing the physical architecture are (1) functional performance, (2) availability and other "-ilities" as achieved through such characteristics as fault tolerance, (3) growth potential and adaptability, and (4) cost. Achieving sufficient functional performance via the development of the physical architecture has been addressed initially in previous sections of this chapter and will be finished in the next chapter during the development of the allocated architecture. Similarly, most of the system-wide (or suitability) factors described in Chapter 6 are often achieved by additional physical resources and associated functionality. Ultimately many of these additional capabilities as

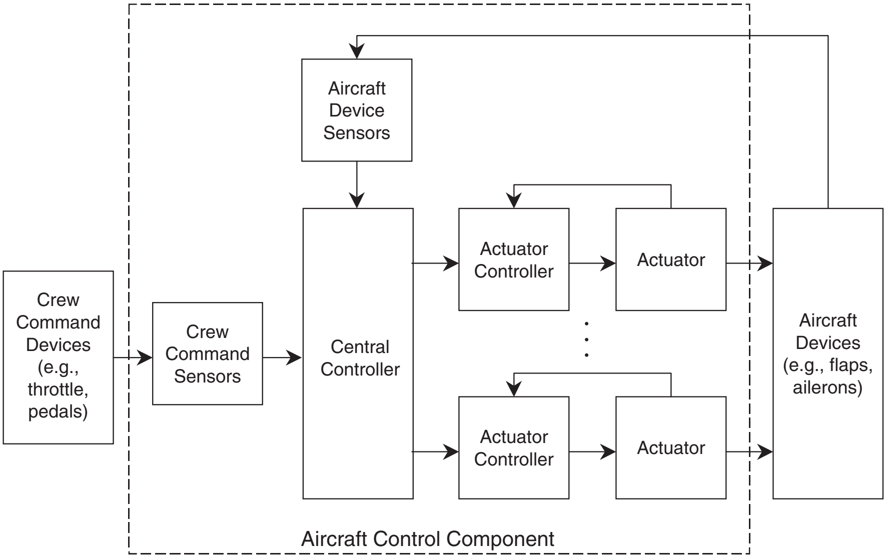


FIGURE 8.7 Block diagram of an aircraft control system.

well as cost are issues of trade offs. These trade offs need to be examined during the evaluation of alternate allocated architectures. Achieving substantial fault tolerance is nearly always important for a system. Finally, there are several issues that impact the ability to grow or adapt a system to changes needed by the stakeholders. The elusive issue of design flexibility is often discussed but difficult to achieve in general. Flexibility is related to such topics as modularity, complexity, and loose versus tight coupling.

Section 8.6.1 addresses the architectural concepts of centralization versus decentralization and distribution of functions and components. Examples from automated systems are used to illustrate these concepts. Section 8.6.2 discusses some new ideas for design flexibility. Section 8.6.3 focuses on the design issues of a physical architecture associated with increasing fault tolerance and availability through redundancy of physical assets, software assets, information, and time.

CASE STUDY: FBI FINGERPRINT IDENTIFICATION SYSTEM

Since the advent of modern information processing technology the Federal Bureau of Investigation (FBI) has sought ways to improve and perfect its fingerprint collection, identification, and archival systems. By 1993 the Bureau's Integrated Automated Fingerprint Identification

System (IAFIS) consisted of three major interactive segments: the Identification Tasking and Networking (ITN/FBI) segment, the Interstate Identification Index (III/FBI) segment, and the Automated Fingerprint Identification System (AFIS/FBI) Segment. In 1993 proposals were solicited from industry to address the ITN/FBI segment.

Among the many challenges associated with developing a competitive technical solution was the subset of requirements related to processing the fingerprint images. Fingerprint images arrive at the FBI through several means. The most common is the widely recognized set of impressions made on a paper form known as a ten-print card. Since the majority of cards comply with a standard set of dimensions, it is a straightforward matter to determine the expected size of the binary image file created when the cards are processed by a digital scanner; both the front and the back sides are scanned.

The following discussion is concerned with the decompression of the scanned card image, followed by its presentation to an expert fingerprint analyst for classification and identification. The FBI's request for proposal (RFP) included a detailed specification for the segment and all sub-elements including the ten-print processing subelement (TPS). According to the RFP the TPS would consist of workstations organized into workgroups. Each workgroup would thus be analogous to one of the many FBI teams engaged in fingerprint analysis. Typically a team consists of a supervisor and perhaps a dozen expert fingerprint analysts. The supervisor's role is to manage the classification and identification of the numerous fingerprint card submissions that the FBI handles on a daily basis. The specification also quantified specific processing requirements for the daily influx of ten-print cards, which at the time of the RFP were given to be an average of 30,000 per day. For example, all incoming cards were required to be scanned and converted to binary data so that they could be distributed electronically to the finger print analysts for subsequent processing. To minimize any impact to the communications infrastructure, the specification required that the images be compressed at a ratio of 10 to 1 prior to transmission over the local area network.

Data concerning the processing response time demands on the fingerprint analysts were also included within the RFP. Chief among the critical task processing times are (1) the average time for the analyst to perform a fingerprint image comparison (FIC), given as 60 seconds, and (2) the time allowed for the display of the human-machine interface screen, including fingerprint images, given as 1 second from the time of the request. Thus the average processing time that a fingerprint analyst requires to complete the task associated with an individual ten-print card was taken to be 60 seconds. This meant that the component performing the decompression function needed to be fast enough to sustain an input queue of ready and available images for each fingerprint analyst.

A second complicating fact was the decompression algorithm. At the time the RFP was released, the most popular algorithm available was based upon a high-quality wavelet scalar quantization (WSQ) approach. The popularity was based on common knowledge among the bidders that the National Institute of Science and Technology (NIST) was about to revise the algorithm specification in preparation for a formal certification. Public access to the algorithm specification enabled the competing design teams of the ITN/FBI segment to benchmark an implementation of the WSQ algorithm in order to quantify its processing requirements. In general the implementations were found to be floating-point arithmetic intensive. As a result it was recognized that such execution behavior is well suited to the latest family of high-performance machines known as reduced instruction set computers (RISC). The specific implementation could be either a software routine or a custom-fabricated large-scale integration (LSI) chip impeded into a math coprocessor card. See Figure 8.8 for a flowchart illustrating the six decision options with an associated block diagram for each option.

Based upon the data provided in the RFP, performance data collected from benchmarks of competing decompression algorithms, and performance data collected from the manufacturers of the computer hardware proposed to host the algorithms, a trade study was conducted to determine how to best implement the function. The particular study described here analyzed six alternate allocations for decompressing the fingerprint images:

a. Implement in software on the workstation within each work group by increasing the TPS workstation processing capacity to enable all decompressions to be performed locally on the individual analysts' workstation.
b. Implement in software on the work group's server by increasing the TPS servers processing capacity to enable all or some decompression to be performed locally on the TPS server for a given work group.
c. Implement in software by distributing the decompression among under-utilized workstations and server processors enterprise-wide, without having to increase the total number of processors or their inherent processing capacity.
d. Implement in software by distributing the decompression among under-utilized workstations and server processors on each local network, without having to increase the total number of processors or their inherent processing capacity.
e. Implement in hardware on the workstation by adding a WSQ coprocessor card in all TPS workstations to perform the decompressions locally.

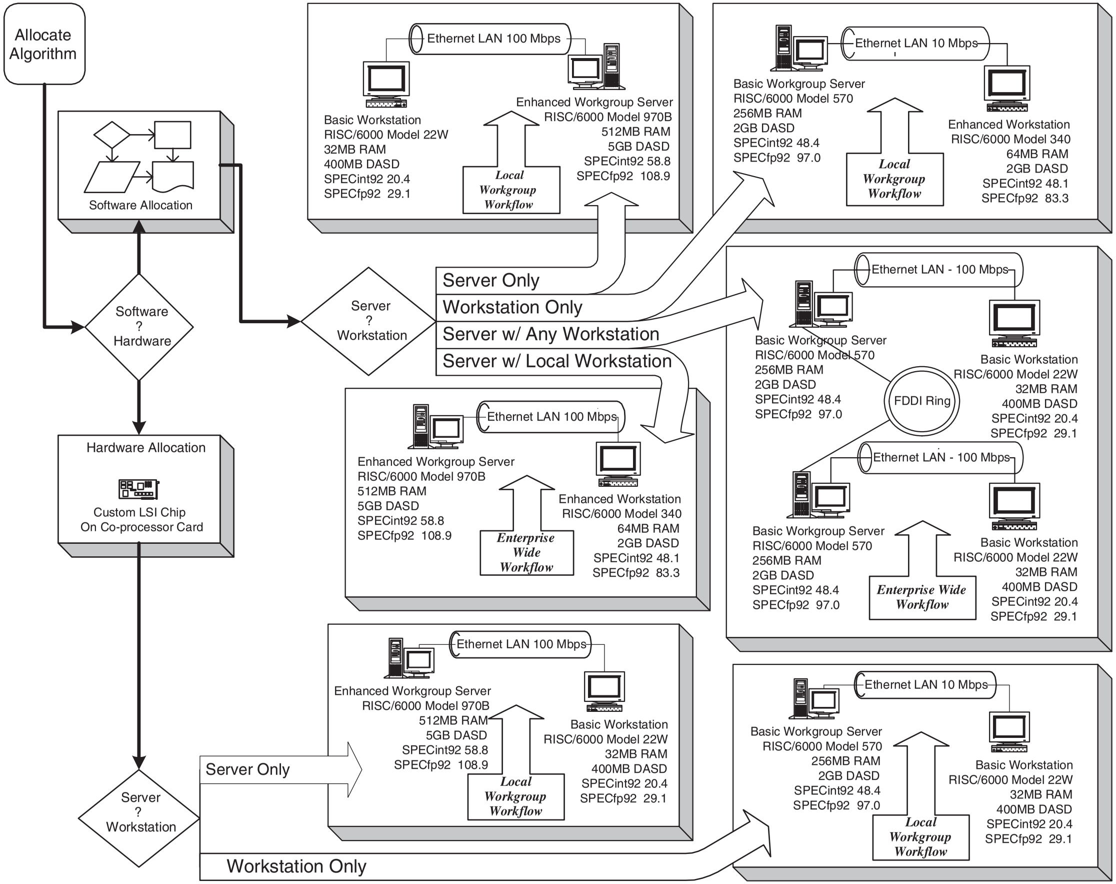


FIGURE 8.8 Flow chart of alternate functional design allocation options with associated block diagrams.

f. Implement in hardware on the server by adding a WSQ coprocessor hardware card in all TPS servers to perform all or some of the decompressions.

The bidder on the basis of a thoughtful process developed the set of six alternatives in Figure 8.8.

Table 8.5 shows a morphological box that contains these six options, as well as many other possibilities.

The first row shows the generic components that were part of this segment, as shown in Figure 8.8. The second through fourth rows show possible instantiations of the generic components. The six alternatives defined for the trade study shown on the previous page are designated with the letters a, b, c, d, e, and f at the bottom of each box in the matrix.

The result of producing this morphological box suggested some new alternatives that would have been competitive with the six analyzed in the trade study; these are shown as g and h in Table 8.5.

Provided by Tim Parker

8.6.1 Major Concepts for Physical Architectures

Nearly every physical architecture is either centralized or decentralized. A *centralized architecture* uses a central location for the execution of the transformation and control functions of the system. A *decentralized architecture* has multiple, specific locations at which the same or similar transformational or control functions are performed. The block diagram for an aircraft control system in Figure 8.7 shows a decentralized architecture; note that there is a central controller, but the controllers for each of the aircraft's actuated devices have been decentralized. In the decentralized architecture shown in Figure 8.7, the central controller manages the decentralized device controllers. A centralized architecture would not have the individual device controllers; rather, the centralized controller would perform all of the functions.

A *distributed architecture* is one in which there are two or more autonomous processors connected by a communications interface and running a distributed operating system [Coulouris et al., 1994; Shuey et al., 1997]. The distributed operating system enables the processors to coordinate their actions and share the system's resources. The processors can perform the same functions, depending upon the needs of the system. Processing control issues for a distributed system are handling the redistribution of processing functions after partial failures; managing moves, changes, and additions to the processing activities; and synchronizing processing activities to meet performance and efficiency objectives. An important distinguishing feature of a distributed system architecture is that the users are unaware of the distribution of processing.

TABLE 8.5 Morphological Box for the Card Image Decompression Component

Workstation	Server	Software	LSI Chip	Workflow Management	Communications
Basic Workstation RISC/6000 Model 22W 32MB RAM 400MB DASD SPECint92 20.4 SPECfp92 29.1 (b, c, e, f) (g, h)	Basic Server RISC/6000 Model 570 256MB RAM 2GB DASD SPECint92 48.4 SPECfp92 97.0 (a, c, e) (g, h)	No WSQ Algorithm (e, f) (g, h)	None (a, b, c, d)	Local Workgroup Workflow (a, b, d, e, f) (g)	Ethernet LAN (10BaseT)—10 Mbps (a, e)
Enhanced Workstation RISC/6000 Model 340 64MB RAM 2GB DASD SPECint92 48.1 SPECfp92 83.3 (a, d)	Enhanced Server RISC/6000 Model 970B 512MB RAM 5GB DASD SPECint92 58.8 SPECfp92 108.9 (b, d, f)	WSQ Algorithm (a, b, c, d)	WSQ on LSI Chip (d, e) (g, h)	Enterprise Wide Workflow (c) (h)	Ethernet LAN (100BaseT) – 100 Mbps (b, d, f) (g)
					FDDI WAN—100 Mbps (c) (h)

A distributed system can be either homogeneous or heterogeneous. The earliest distributed systems were *homogeneous*, that is, comprised of identical processors, running identical operating system and application software, and connected via a single communications network. Users on a homogeneous distributed system view the system as their processor but obtain the benefits of being able to share data with each other over wide geographic regions. Eventually some processors become much busier than others and the issue of load sharing arises; load sharing distributes computational tasks from one processor to another. Note *load sharing* is the reallocation of functions to different resources in the physical architecture and is therefore an issue in the allocated architecture. Load sharing causes users to access and share multiple processors and provides increased response times in many cases. Finding the best approach to load sharing is quite complex.

Heterogeneous distributed systems have two or more types of processors comprising the processor network, plus operating and application software and one or more communications networks connecting the processors. The Internet is the most common example of a heterogeneous distributed system. Specially designed, heterogeneous distributed systems are, or will, enable medical support in hospitals by both specialists and generalists, financial transactions, fingerprint analysis by both experts and automated assistants, review of tax records by both experts and automated assistants, and analysis of data collected by satellites by a wide variety of researchers. Each architecture shown in Figure 8.8 for the FBI fingerprint identification system case study is a heterogeneous network involving two types of processors, clients and servers.

The major reasons that a distributed processing architecture is attractive in designing systems are transparency, openness, scalability, resource allocation, concurrency, and fault tolerance. *Transparency* means that the users view the distributed system as a complete system, *without* any knowledge of how the hardware and software components are performing. An *open architecture* is one for which the hardware and software interfaces are sufficiently well defined so that additional resources can be added to the system with little or no adjustment. *Sealability* means that multiple-sized versions of the system are available. *Resource sharing* exists when more than one hardware and software module can be used to execute the same task with no human intervention. A *concurrent architecture* is one in which multiple tasks are being executed simultaneously. A single processor can perform concurrent operations by interleaving the operations of multiple tasks; however, multiple, distributed processors can clearly perform concurrent operations without any direct knowledge of what the other processors are doing. Finally, *fault tolerance* is achieved if the distributed system can adjust its operations when one of the hardware or software elements fails. Details for achieving fault tolerance are discussed in Section 8.6.2.

A client–server architecture is a software architecture that is superimposed on a distributed system to facilitate processing and management of the system. The *client–server architecture* distinguishes between client processes (requestors) and server processes (task completors). Each distributed processor is performing its assigned task; when one processor needs support from another processor, the processor needing support becomes a client and issues a request across the network. The processor that accepts the request becomes the server, responds that it will complete the request, and uses both hardware and software resources to complete the task and send the result to the client. Note this server may have just issued a client request of its own and may be waiting for a response from some other processor. Servers may be set up for database, file, print, fax, mail, communication, and imaging operations. This client–server architecture will be discussed in more detail in Chapter 10.

8.6.2 Design Flexibility

Many engineers talk and write about design flexibility, modularity, loose coupling, complexity and other such topics, but it is usually quite difficult to find nuggets that prove useful in the real world. This section will explore some of these ideas.

In Chapter 6 we talked about how much change occurs during the design process and how this change makes success elusive. In addition, most systems are designed to last many years or even decades. The mark of a long-lived system is one that has been upgraded successfully many times. These many upgrades are only possible if the system's architecture has provided an adaptable platform for such upgrades. The Sidewinder missile of the U.S. Navy and Microsoft's Windows NT operating system are two examples of architectures have supported dramatic changes over many upgrades, such that the original design is no longer present but the "architecture" remains. So in addition to working hard to keep track of the changes that are occurring in the requirements, we can also design our systems to be more "changeable" in the future.

Fricke and Schulz [2005] address this problem by defining four aspects of changeability: flexibility, agility, robustness, and adaptability.

- "*Robustness* characterizes a systems ability to be insensitive towards changing environments. Robust systems deliver their intended functionality under varying operating conditions without being changed (see Taguchi [1993] and Clausing [1994]). That is, no changes from external to be implemented into such systems to cope with changing environments.
- *Flexibility* represents the property of a system to be changed easily. Changes from external have to be implemented to cope with changing environments.
- *Agility* characterizes a system's ability to be changed rapidly. Changes from external have to be implemented to cope with changing environments.
- *Adaptability* characterizes a system's ability to adapt itself towards changing environments. Adaptable systems deliver their intended functionality

> under varying operating conditions through changing themselves. That is no changes from external have to be implemented into such systems to cope with changing environments."

Some examples of each of these should help make the points emphasized by Fricke and Schulz. An all-terrain automobile such as a jeep might be an example of a robust vehicle; it can travel reasonably well on many different surfaces. If this all-terrain vehicle can also have a cloth top that can removed and stored, this adds to its robustness. A flexible system is one that can interface easily with many other types of systems, each of which might be changing. For example, laptop computers with many USB ports in the 2007 time frame can interact with nearly all printers, projectors, and control devices. The peripherals or other systems that can plug into the USB ports still have to be changed as the environment changes, but the core computer does not need to change for these reasons. Flexibility is important for future upgrades. An agile system is designed to be changed rapidly. Here a race car comes to mind. Race cars have to be modified dramatically to run well on different race tracks from one week to the next. A great deal of money is spent on the design to facilitate these rapid changes. Adaptable man-made systems are being designed but with some limitations. Microsoft has designed its operating and office products to learn and adapt to different users so as to facilitate the performance of these different users. While this has been the goal at Microsoft, many feel (including this author) that their efforts are far from successful.

Fricke and Schulz [2005] describe three basic design principles that support all four types of design for changeability and six extending design principles, each of which supports a subset of the types of design for changeability. The three basic principles are ideality/simplicity, independence, and modularity/encapsulation. The six extending principles are integrability, autonomy, scalability, non-hierarchical integration, decentralization, and redundancy. Aspects of decentralization were discussed above. This next section addresses redundancy for fault tolerance, a form of adaptability.

8.6.3 Use of Redundancy to Achieve Fault Tolerance

Fault tolerance was discussed in Chapter 7 from the perspective of functions that need to be performed to detect errors, confine the damage, recover from the damage, isolate the damage, and report the problem. Design issues associated with the physical architecture are just as important in achieving fault tolerance. A primary source of high availability and fault tolerance is redundancy. Often hardware redundancy receives most of the attention. However, Johnson [1989] identified four elements of *redundancy:* hardware, software, information, and time. *Hardware redundancy* uses extra hardware to enable the detection of errors as well as to provide additional operational hardware components after errors have occurred. This hardware redundancy can be implemented in passive, active, and hybrid forms.

Passive hardware redundancy masks or hides the occurrence of errors rather than detecting them; recovery is achieved by having extra hardware available when needed. The rest of the system and its operators are commonly not even aware that an error has occurred. This approach only works as long as there are sufficient hardware replicas to continue to mask errors. The most common passive implementation is called triple modular redundancy (TMR) and relies on a majority voting scheme to mask an error in one of the three hardware units. Figure 8.9 (top left) shows TMR; unfortunately the single "voter" element is a single point of failure in this system. Therefore TMR is often implemented as triplicated TMR (Fig. 8.9 bottom right). Triplicated TMR implements three voters and produces three versions of the output, which are usually sent to another module that has been implemented as triplicated TMR. Naturally, there is nothing magical about three; N-modular redundancy (NMR) is the generalization of TMR. TMR can mask a single error; 5-MR can mask two errors, etc.

Voting is a common conflict resolution technique used inside a computer, as well as with groups of people. However, implementing voting inside a system has some unexpected difficulties. Issues in voting implementation are establishing the time at which the computation was done, the precision of numbers achievable in a digital computer, and the need to produce a single answer eventually. Timing of the computations is critical because the hardware and software components producing inputs to the *voter* may be performing repetitive computations on a data stream and be out of synchronization. For repetitive operations there must be some synchronization mechanism involved to ensure that the vote is being taken on computations from the same samples of data stream of inputs.

The precision issue addresses the concern that there is some imprecision in numerical operations involving digital equipment. Quantization of a number

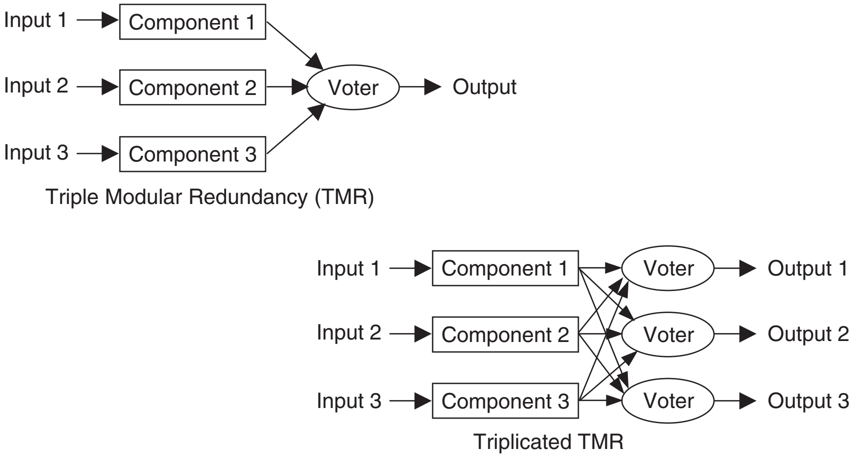


FIGURE 8.9 TMR and triplicated TMR (after Johnson [1989]).

on a digital computer can produce several different valid results. As a result the voter may see three different outputs from the three components, but the outputs are the result of normal processing operations. In many cases the majority voting scheme is replaced with either a selection of the median value or truncation of the numerical values to some predefined level of significant digits.

The last issue, the production of a single answer, requires that a single point of failure be introduced. When the final result (e.g., bank account balance or control signal to the rudder) has to be delivered by the system in question, this final answer is determined on a single processor.

Finally, voting for passive redundancy can be achieved via hardware or software. A hardware implementation is faster but usually requires more cost, space, power, and weight. A software implementation (see Figure 8.10) provides greater flexibility for change but can also require additional cost, space, power, and weight in the form of processors if voting is a major part of the system's redundancy, which is often the case.

Active hardware redundancy attempts to detect errors, confine damage, recover from the errors, and isolate and report the fault, as described in Chapter 7. The basic building block for active hardware redundancy is called *duplication with comparison*; see Figure 8.11 for a hardware implementation. Two identical units are used to compute the same output for the same set of inputs; these outputs are compared in a "comparator." If the outputs disagree by a predefined amount, an error is declared. (Note the issues of synchronization and precision also apply here.) Once an error is declared, functionality to confine the damage, recover from the errors, and isolate the reports is activated.

Hot and cold standby sparing are different than duplication with comparison and are the most common approaches to active redundancy; see Figure 8.12. In *hot standby sparing* multiple replicas of a component are performing identical functions; only one of them is providing outputs, but all are ready to take over with no delay. Error detection in standby sparing is not done by comparing outputs from redundant components, but by examining the output for known errors or monitoring the component for inactivity. A watchdog timer is an

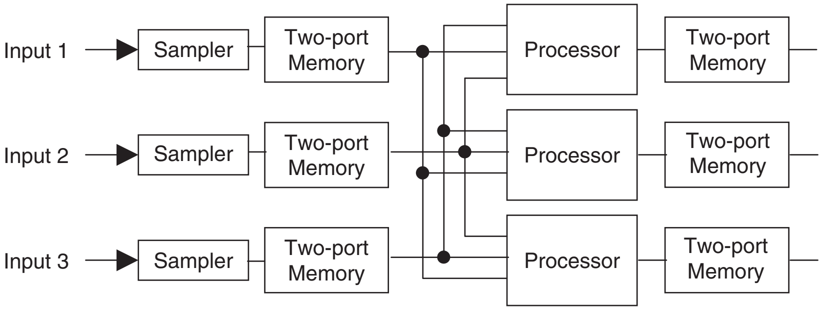


FIGURE 8.10 Software implementation of voting for triplicated TMR (after Johnson [1989]).

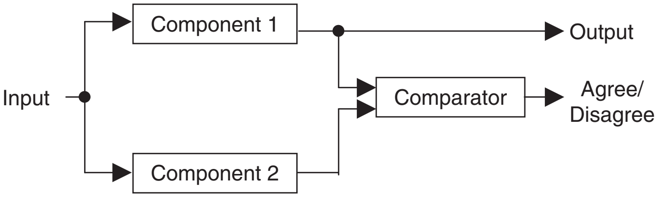


FIGURE 8.11 Hardware duplication with comparison (after Johnson [1989]).

example of this latter approach; a *watchdog timer* declares a fault if it is not continuously reset by the component with which it is associated.

Cold standby sparing maintains the component replicas in a nonoperational mode until needed. This is useful for applications where short disruptions are acceptable or long life is key, for example, spacecraft operations. For real-time applications, hot standby sparing is critical to success but increases power consumption and decreases the life of the system. Standby sparing is most commonly used by providing multiple, excess processors, any of which can be used to perform necessary system functions. When one processor fails, a controller no longer assigns tasks to that processor, with the slack being absorbed by the remaining processors.

The final example of active hardware redundancy, *pair-and-a-spare*, combines the features of duplication with comparison and standby sparing. Figure 8.13 shows a comparison (far right) of the outputs of two active, identical components to detect an error. If the comparison yields a disagreement, the "*N* to 2" switch is directed to select alternate components for conducting the comparison. Note the error detection logic from standby sparing; is also present.

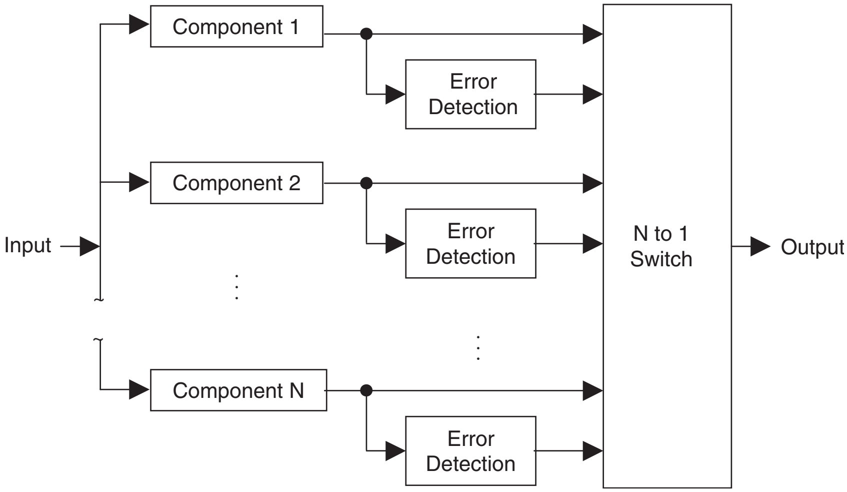


FIGURE 8.12 Standby sparing with N-1 replicas (after Johnson [1989]).

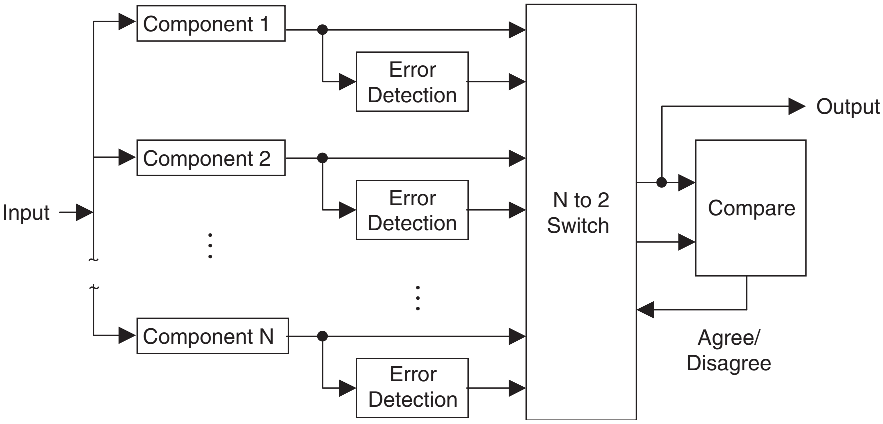


FIGURE 8.13 Pair-and-a-spare active hardware redundancy (after Johnson [1989]).

Examples of *hybrid hardware redundancy* are the combination of N-modular redundancy with spares, and the triple-duplex architecture, which combines TMR with duplication with comparison. Critical computation systems usually use passive or hybrid redundancy. Systems that have requirements for long life and high availability without critical computations employ active redundancy. Active redundancy is usually less costly; hybrid redundancy is the most costly.

Software redundancy is a second means for detecting and recovering from errors. N-version software redundancy is a seldom-used approach to provide multiple operational software components in the event of a software failure. Each version is programmed by separate groups of programmers, assuming that while each group may make mistakes, no two will make the same mistake. More common forms of software redundancy are consistency and capability checks; both can be used for error detection in standby sparing. *Consistency checks* compare the output of a component with known characteristics of that output, for example, minimum and maximum values. *Capability checks* are software designed to run periodic hardware tasks with known answers.

Information redundancy is achieved by adding extra bits of information to enable error detections using special codes [Johnson, 1989]. Information redundancy is useful to catch *system*-induced errors rather than component faults; however, system-induced errors can be indicative of component faults if the errors occur with sufficient frequency. Information redundancy is a very rich area, having many alternate approaches. Information redundancy is one form of error detection that can be used for standby sparing; see Figure 8.12.

Time redundancy can be used to replace hardware and software in non-real-time systems to achieve error detection. When extra processing time is available, computations can be *performed* multiple times with a single hardware and software combination and compared. If discrepancies exist, an error has been detected. This approach is also used for error detection in standby systems and is quite useful in distinguishing between transient and permanent errors. Time redundancy assumes that additional time exists for functional performance to enable the needed error detection and recovery. On the plus side, time redundancy can save significantly on hardware and software, reducing cost, weight, power, and other key suitability issues.

8.7 SUMMARY

The focus of this chapter has been the resources that comprise the system, called the physical architecture. The system is first segmented into its top-level components; the segmentation progresses down to the configuration items (CIs), or hardware and software elements, facilities, people, procedures, and user's manuals.

The physical architecture can be either generic or instantiated; the generic physical architecture is an abstract separation of the system's resources into components before any key performance decisions are made. The instantiated physical architecture specifies the performance characteristics of each element of the generic physical architecture to the degree needed for performance modeling of the system.

Creativity techniques are important to aid the generation of alternate, instantiated physical architectures. The morphological box was described in detail and illustrated as an effective technique for gathering creative ideas and increasing the chances of combining these creative ideas into a sound, instantiated physical architecture. The morphological box is defined by the generic physical architecture and then provides slots for alternate ideas for instantiated physical components of each segment.

Representing the physical architecture using a block diagram was presented in this chapter. Block diagrams are completely non-standardized representations of the system's components, showing the major flows of electromechanical energy between the components.

Finally, key concepts, such as centralized and decentralized and distributed and client–server architectures were presented. The decentralization of transformation and control functions and the distribution of functional and physical elements of the architecture have become the norm in most system's architectures. These concepts were defined and illustrated.

Redundancy in hardware, software, information, and time was presented since achieving fault tolerance is often a critical design issue that the engineer of the system must address. Hardware redundancy is the most commonly discussed and implemented approach to achieving fault tolerance with the physical architecture. Software redundancy is almost always too expensive to develop. Information redundancy, adding extra bits to data elements for the purpose of checking the meaningfulness of data elements later, is used extensively on communications interfaces that become part of the physical architecture. Utilizing unused data processing time to repeat computations, time redundancy, is not a common approach.

CASE STUDY: COMMERCIAL AIRCRAFT CRASH AT SIOUX CITY, IOWA

On July 19, 1989, United 232 (a DC-10 aircraft) crashed into a corn field next to the Sioux City airport in Iowa while trying to make an emergency landing after losing one of three engines. In all, 110 passengers and one flight attendant were killed during this emergency landing; 185 people survived the accident, some without a scratch.

Engine failure is the most commonly trained maneuver in simulators. The DC-10 has three engines; one on each wing and one on top of the fuselage in the vertical tail (or horizontal stabilizer). United 232 lost the engine on top of the fuselage due to the loss of a fan disk; the fan disk separated from the engine and crashed through the tail. Pilots fought through the engine loss by porpoising (rotating the thrust levels) the two remaining engines to land in Sioux City. However, the descent rate of the landing was too great; the aircraft caught fire upon landing, tumbled, and broke apart in corn and soybean fields.

The fan disk, about 300 pounds of titanium, on the number two engine was missing; it had shattered into pieces and crashed through a chamber designed to contain such a break-up. There are three independent hydraulic systems on the DC-10 aircraft; a unique engine powers each hydraulic system. The hydraulic system on an aircraft provides the forcing function for the aircraft's stabilization systems: the ailerons on the wings that permit the aircraft to bank right and left, the rudder that allows the aircraft to turn right and left, the elevators on the tail that cause the aircraft's nose to rotate up or down, and the flaps and slots on the wings that permit the aircraft to change the amount of lift generated by the wings. Losing engine number two should have only caused the loss of one of the three hydraulic systems. However, the three independent hydraulic systems converge in the tail at the exactly the location that the fan disk ripped out, the single point of failure for all three hydraulic systems.

Experts believe there was a preexisting fracture on the fan disk. Ultrasonic sensors are used to detect fractures during production. However, these sensors do not provide good results when the fracture is near the surface. The National Transportation Safety Board (NTSB) investigators concluded that the fracture had been there since the fan disk was built. The fracture would have grown with use; the maintenance crew was blamed for not finding the fracture during routine maintenance activities. Nonetheless, this does not dismiss the design flaw of a single point of failure for what were considered to be three redundant hydraulic systems [Magnuson, 1989; Birnbaum, 1989].

PROBLEMS

8.1 Create a generic physical architecture for the ATM problem in Chapters 6 and 7. Create a morphological box for your generic physical architecture of the ATM. Identify three instantiated physical architectures based upon the morphological box.

8.2 Create a generic physical architecture for the OnStar system in Chapters 6 and 7. Create a morphological box for your generic physical architecture of OnStar. Identify three instantiated physical architectures based upon the morphological box.

8.3 Create a generic physical architecture for a personal computer. Create a morphological box for your generic physical architecture of a personal computer. Identify three instantiated physical architectures based upon the morphological box.

8.4 Create a generic physical architecture for a stereo system. Create a morphological box for your generic physical architecture of a stereo system. Identify three instantiated physical architectures based upon the morphological box.

8.5 Create a generic physical architecture for the development system of an air bag system. Create a morphological box for your generic physical architecture of the development system. Identify three instantiated physical architectures based upon the morphological box.

8.6 Create a generic physical architecture for the manufacturing system of an air bag system. Create a morphological box for your generic physical architecture of the manufacturing system. Identify three instantiated physical architectures based upon the morphological box.

8.7 Using the information in Figure 8.7 create a block definition diagram and an internal block diagram for the "Aircraft Control Component," which is inside the dotted lines of the figure. Be sure to use the semantics and syntax of SysML. Note: You will have to ignore any arcs coming from or going to components outside the dotted line.

8.8 You are on the elevator design team and have just convinced the team that the block decomposition at the subsystem level (Figure 3.14) is incorrect. You have convinced the team to add a communications bus so that the communications between the subsystems can be more efficiently routed through the communication bus. Modify the block definition diagram and internal block diagrams shown in Figures 3.14 and 3.16, respectively, for the elevator subsystems to show this design change. Consider the communications bus to be a new component or subsystem.