Chapter 9

Allocated Architecture Development

9.1 INTRODUCTION

The development process for the allocated architecture is the activity during which the entire design comes together. The allocated architecture integrates the requirements decomposition with the functional and physical architectures. The process of developing the allocated architecture provides the raw materials for the definition of the system's external and internal interfaces and is the only activity in the design process that contains the material needed to model the system's performance and enable trade-off decisions. The reader should not infer from this discussion that the requirements development is started and finished, followed by the functional architecture, followed by the physical architecture, followed by the allocated architecture. Rather, the design process is like peeling an onion; each of these activities in the design process should be completed at a high level of abstraction (low level of detail), culminating in an allocated architecture at this high level of abstraction for a set of subsystems that comprise the system. Then the entire process is repeated at a lower level of abstraction (greater detail) for the next tier of components (peel of the onion), consistent with the Vee model discussed in Chapter 1. This repetition at lower and lower levels of abstraction (greater and greater detail) is continued as long as useful to the design process. As details determine problems with the design, decisions are reviewed and changes are implemented at the higher levels of abstraction as needed.

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

This chapter describes the activities involved in developing an allocated architecture in detail: allocate functions to subsystems; trace non-input/output requirements and derive requirements; define and analyze functional activation and control structure; conduct performance and risk analysis; document architectures and obtain approval; and document subsystem specifications.

The methods introduced in this chapter match the functions that comprise the development of the allocated architecture. Various methods are discussed for allocating functions of the system in question to subsystems and components of the system. The derivation of input/output, system-wide and technology, trade-off, and qualification requirements is discussed as a key method for providing the material to complete the component specification. Three methods for flowing down system-wide and technology requirements that have been traced to the system are described. Models for defining and analyzing functional activation and control structures are discussed in Chapter 12 and are therefore not presented in this chapter. However, critical system-wide issues associated with functional activation and control are discussed here. A normative model for conducting trade studies and risk analyses is presented in Chapter 13. Examples of common trade studies and risk analyses are discussed and illustrated in this chapter. No new models are introduced in this chapter.

The exit criterion for finishing the allocated architecture is the acceptance of the design by the stakeholders. The acceptance of the design by the stakeholders should involve a detailed understanding that the requirements development process has met the major characteristics of the requirements, as defined in Chapter 6: thorough understanding of how the allocated architectures of the systems in each life-cycle phase will meet the requirements as defined, belief that the design trades have accurately reflected the trade-off requirements, and agreement that the test or qualification systems in each phase of the life cycle are adequate for qualification requirements as defined.

9.2 OVERVIEW

The *allocated architecture* provides a complete description of the system design, including the functional architecture allocated to the physical architecture, derived input/output, technology and system-wide, trade off, and qualification requirements for each component, an interface architecture that has been integrated as one of the components, and complete documentation of the design and major design decisions.

There are five major activities associated with the development of the allocated architecture:

- Allocate functions and system-wide requirements to physical subsystems
- Allocate functions to components
- Trace system-wide requirements to system and derive component-wide requirements

- Define and analyze functional activation and control structure
- Conduct performance and risk analysis
- Document architectures and obtain approval
- Document subsystem specifications

Figure 9.1 shows these five functions in an IDEF0 (Integrated Definition for Function Modeling) diagram for developing the allocated architecture; see Appendix B for the full model. Note that Sections 9.3 and 9.4 address the two subfunctions under the first function (these were combined to make the diagram easier to read). As can be seen by the flow of information among these activities, substantial interaction and feedback is required among the first four to make sure the design works; this feedback and control was discussed in Chapter 7. However, viewing the development of the allocated architecture in isolation would be inappropriate. The developments of the three architectures (functional, physical, and allocated), which we have been discussing, all have to proceed in parallel because insight or changes in one have repercussions in the others. Figure 9.2 puts the allocated architecture development in context with the other architectures and requirements development.

As discussed in the introduction, the design process proceeds through the steps shown in Figure 9.2 several times, at decreasing levels of abstraction. The more complex the system's functionality and tightly coupled the system's components are, the more important is the repetition of the design process at decreasing levels of abstraction (increasing detail). Initially, the design process establishes functional and physical decompositions, which are united to form the allocated architecture. The allocated architecture divides the design problem into chunks, primarily along the lines of the physical architecture, namely the system's components. Naturally, these design decisions should not be made prematurely; there should be adequate confidence that little or no modifications will be needed. Yet, as the design process evolves through additional repetitions of the activities shown in Figure 9.2, the more detailed simulation models and trade studies may provide justification for modifying earlier design decisions.

The primary benefit of making major design decisions early using models and trade studies built at a high level of abstraction is that these initial decisions are aimed at dividing the design problem into manageable chunks that can proceed concurrently with a reasonable chance of success. Dividing the system's design problem into completely independent chunks is not possible. To accommodate this interaction there must be design interfaces just as there are system interfaces. These design interfaces are part of the development system that is being completed concurrently with the design of the operational system. It is critical that the development system provide the time to review and adjust the design chunks; this time can only be provided if the design process begins at a high level of abstraction. Some engineers argue that this initial peel of the onion should be completed within weeks (6–12) after having written a

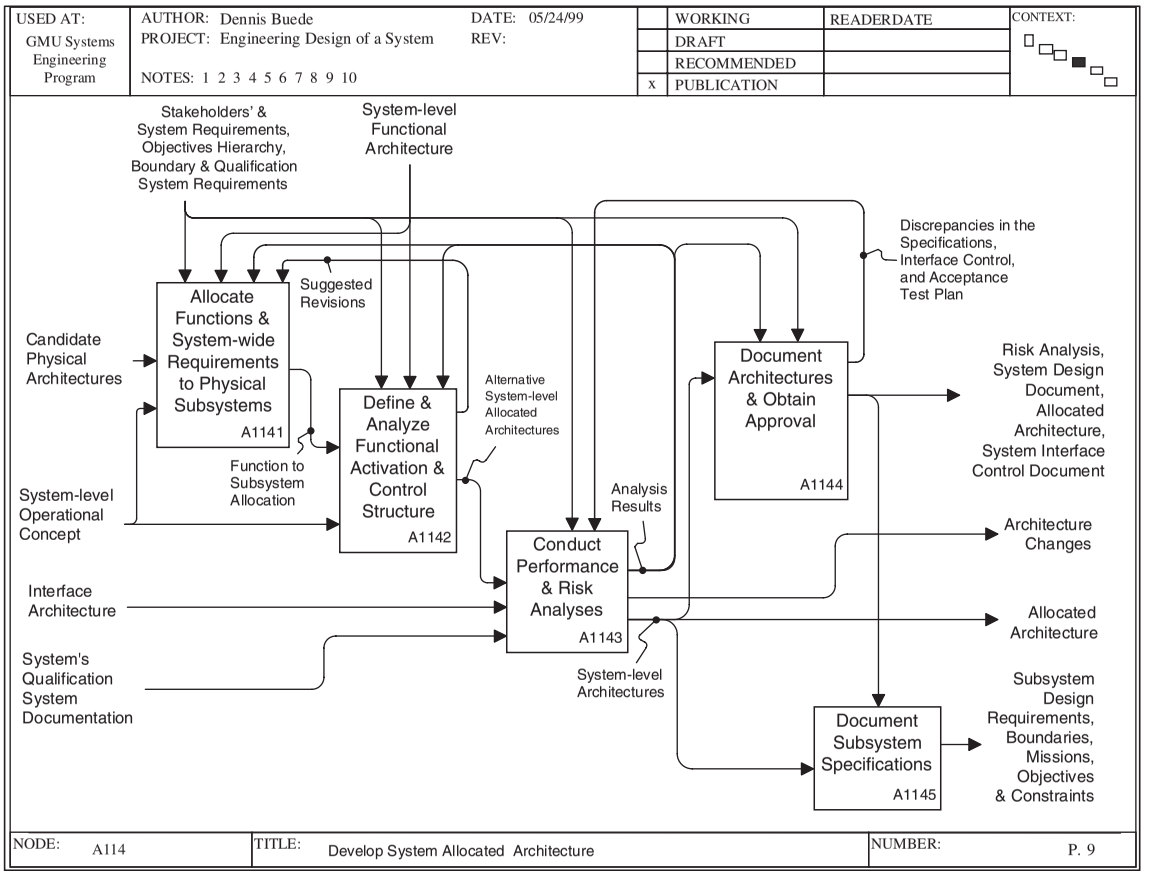

FIGURE 9.1 IDEF0 representation of developing the allocated architecture.

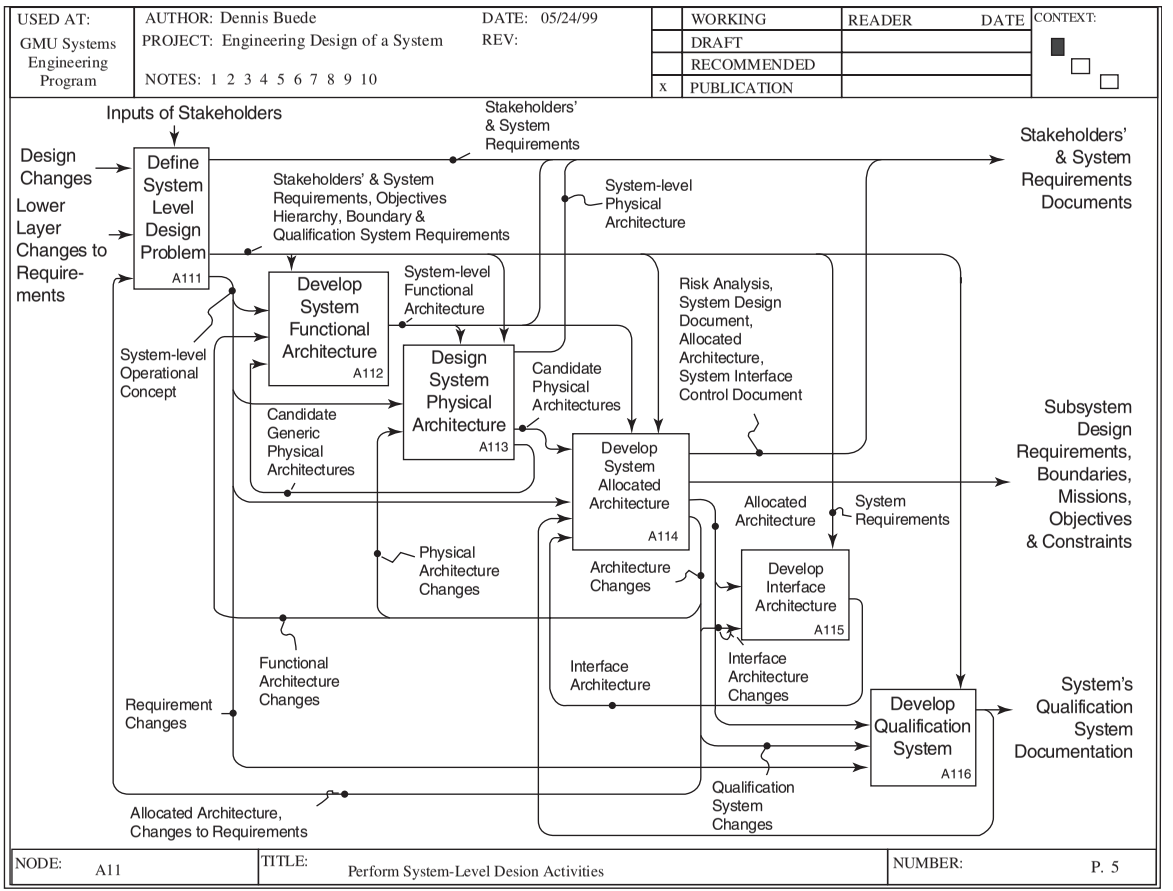

FIGURE 9.2 System-level design activities.

proposal and been awarded a contract. If the design segmentation is not finalized until each component has been decomposed into several levels of detail, there will be no time to adjust this design decision if the division of the system into components is found to be flawed. There is even less chance that the flaws will be found if too many details are analyzed too quickly.

Distinguishing between good decisions and good outcomes is important. If we were in complete control of our environment, then decisions and the outcomes associated with the decisions could be equated. However, as discussed in detail in Chapter 13, decisions must be made in the face of uncertainty with incomplete information and inadequate control of the outcomes. Therefore, saying that a decision was good or bad because the outcomes associated with that decision were good or bad, respectively, is illogical. A decision can be considered good if the people with the best knowledge and largest stake in the decision were involved in the decision, and these people did discuss the relevant alternatives, values, and facts with clarity.

As an example, Ford Motor Company designed and introduced the Edsel in 1957. The Edsel had a large, elongated "0" built into the middle of the grill at the front of the car that caused many people to react negatively on an artistic basis. The Edsel was a complete failure at least partially because the automobile industry was in a recession in 1957 and 1958. Were the design decisions associated with the Edsel bad? It is not possible to tell without knowing more about what design decisions were made and how the design process was carried out. Seven years after the Edsel's introduction, Ford Motor Company introduced the Mustang, which has been a fantastically successful car and has achieved classic status. Were the design decisions associated with the Mustang good? Again, it is not possible to tell without knowing more about them. With time it is much easier to tell whether the outcomes associated with a decision are good or bad, but it becomes more and more difficult to tell whether the decisions that were made were good or bad, especially if those decisions are not documented.

9.3 ALLOCATE FUNCTIONS TO COMPONENTS

After the definition of the functional and physical architectures, the systems engineering team must assign functions from the functional hierarchy to the subsystems and components in the physical architecture. When this is done, the first step in defining the allocated architecture is completed. This allocation of functions to components is often the most crucial design decision made by the engineers of the system. Engineers prefer to allocate processing tasks to software if there will be a future need to update the processing algorithms. However, if speed of processing is critical, hardware can perform the computations much faster. Computer manufacturers experiment with moving some processing tasks from hardware to software, but often find that the speed of processing suffers too much and revert to designing hardware for the

processing tasks. Similar issues arise when considering the decision of allocating a function to people within the system or a combination of hardware and software. This allocation decision is discussed in more detail later.

Figure 9.3 expands upon Figure 9.4 for the allocation of the system's functions to subsystems and components. Clearly allowing the allocation decision to be represented as a mathematical relation, and not a function, as shown in the top left of Figure 9.3 is inadequate; there will be some functions that are not allocated to any component and some functions that are being processed by two or more components. Forcing the allocation of functions to components to be represented as a mathematical function, as shown in the top right of Figure 9.3, solves these problems. However, there may be some components with no functions to perform; these components should either be dropped from the system or the engineers should revisit their functional architecture to ensure that the functional architecture is complete. There is also the possibility that some functions will be performed by the same component; there is nothing wrong with this because the functions can be aggregated into a single function. If as expected all of the components are

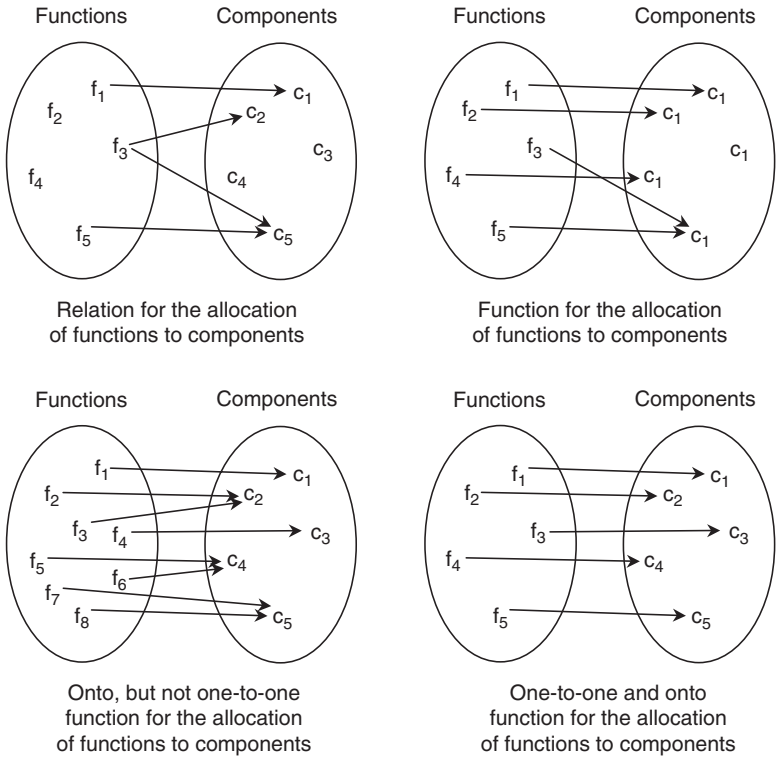

FIGURE 9.3 Mathematical relations and functions for the allocation of engineering functions to components.

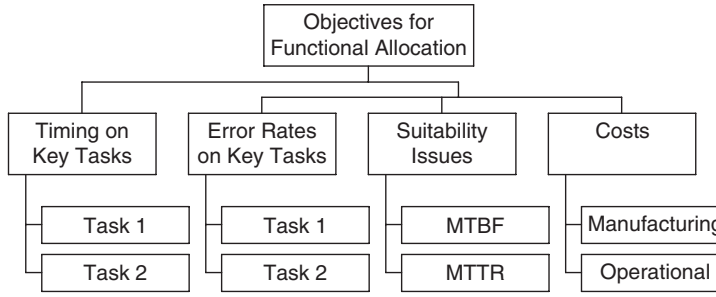

FIGURE 9.4 Sample objectives hierarchy for functional allocation.

needed, the allocation of functions to components will be onto, as shown in the bottom left of Figure 9.3. An onto functional allocation is one-to-one when the number of functions and components is the same, as shown in the bottom right of Figure 9.3.

Note that the mapping of functions to components was picked consciously, rather than the mapping of components to functions. Allowing two components to be mapped to the same function is consistent with the definition of a mathematical function but should be avoided by the engineers of a system. When two components are performing the same function, it will not be possible to segment the responsibilities of the components until the functional and physical architectures are examined in greater detail; this defeats the purpose of iterating through the engineering process as suggested by the Vee model and most engineers of systems.

9.3.1 Define the Allocation Problem

For any single physical architecture and the associated functional architecture, there are many possible allocated architectures that could be defined. The basis on which this allocation is done could be formulated as a multi-objective optimization problem:

1. Maximize the fundamental objective (must be based upon analysis using the fundamental objectives hierarchy). Note that besides common operational performance parameters there are often other elements of the fundamental objectives concerning performance in other phases of the life cycle (for example, maintenance, deployment, and refinement) about which to be concerned.
2. Minimize the number and complexity of interfaces. This is often called *modularization*, which is nearly synonymous with maximizing the ability to encapsulate the functions inside the physical entities of the system. By encapsulation we mean the ability to hide the implementation details of

performing the entity's functions from the remaining parts of the system. Essentially, the remainder of the system should only need to know the outputs of each entity, not how those outputs are produced. Software engineers call this *information hiding*. The concepts of modularity and information hiding are also highly related to the concept of *coupling*. Many systems and software engineers distinguish between tight and loose coupling. Loose coupling decreases complexity, enables flexibility, but often degrades performance. Wikipedia has a nice description of the many types of coupling found in systems.

3. Maximize early critical testing opportunities so as to give engineers a chance to find and fix problems. This is often considered *risk minimization*. Opposing criteria may minimize risks:
   a. Equalizing risks (difficult requirements) across the physical architecture or
   b. Localizing risks in a single element of the physical architecture (the opposite of equalizing risks)

9.3.2 Approaches for Solving the Allocation Problem

In the 1950s and 1960s the major trade offs addressed by engineers consisted of choosing between the human in the system and the system's combined hardware and software resources for performing certain critical functions. In the 30 to 40 years since systems engineers first grappled with these decisions, systems engineers are still using heuristics to resolve these decisions. The engineering and psychology communities believe that there are certain functions that humans perform better than machines, at least in many situations; there is not complete agreement about what these functions are, for example, pattern recognition functions, improvisation, and adaptation. Similarly, hardware and software combined clearly outperform humans in tasks that require responding quickly to control signals, performing repetitive tasks, and performing many different activities at once. Paul Fitts [1951] was the first to try to systematize these allocation issues by producing what has come to be known as a ''Fitts' list'' and later known as ''Men are better at — machines are better at'' or ''MABA — MABA.'' Fitts' first list is shown in Table 9.1.

Sheridan and Verplanck [1978] developed a taxonomy of 10 possible distribution strategies for allocating the functional responsibility of control between the human and the computational resources of the system. These allocation strategies range from having the human be the planner, scheduler, optimizer, and the like, to taking the human out of the system's functions completely; see Table 9.2. For example, the first distribution in the table puts the entire cognitive load on the human, which reflects automation in the 1960s and 1970s, such as machine tools. Entries 5 and 6 reflect the computer developing suggestions for actions but letting the human have approval or intervention capability; this reflects much of the automation in military systems

TABLE 9.1 Original Fitts List from 1951

Humans appear to surpass present-day machines with respect to the following:	Present-day machines appear to surpass humans with respect to the following:
1. Ability to detect small amounts of visual or acoustic energy. 2. Ability to perceive patterns of light or sound. 3. Ability to improvise and use flexible procedures. 4. Ability to store very large amounts of information for long periods and to recall relevant facts at the appropriate time. 5. Ability to reason inductively. 6. Ability to exercise judgment.	1. Ability to respond quickly to control signals, and to apply great force smoothly and precisely. 2. Ability to perform repetitive, routine tasks. 3. Ability to store information briefly and then to erase it completely. 4. Ability to reason deductively, including computational ability. 5. Ability to handle highly complex operations, i.e., to do many different things at once.

today. Entries 7 through 9 reflect the status quo in autopilots for aircraft and trains.

Now that computer-based systems and embedded computer systems are much more sophisticated and prevalent, the most critical functional allocation decision facing systems engineers often relates to the allocation of a function

TABLE 9.2 A Taxonomy of the Distribution of Responsibility between Human and Computer

1. Human does all planning, scheduling, optimizing, etc., and turns task over to computer merely for deterministic execution.
2. Computer provides options, but the human chooses between them, plans the operations, and then turns task over to computer for execution.
3. Computer helps to determine options, and suggests one for use, which human may or may not accept before turning task over to computer for execution.
4. Computer selects option and plans action, which human may or may not approve, computer can reuse options suggested by human.
5. Computer selects action and carries it out if human approves.
6. Computer selects options, plans and actions and displays them in time for human to intervene, and then carries them out in default if there is no human input.
7. Computer does entire task and informs human of what it has done.
8. Computer does entire task and informs human only if requested.
9. Computer does entire task and informs human if it believes the latter needs to know.
10. Computer performs entire task autonomously, ignoring the human supervisor who must completely trust the computer in all aspects of decision-making.

between hardware and software. Allocating a function to hardware has the benefit of reduced development cost and faster processing and response time. The advantages of allocating to software are the flexibility to modify the function in the future as design problems are found or new algorithms prove superior in terms of timing, quality, or quantity measures.

Price [1985] developed the principles (Table 9.3) for functional allocation that are primarily related to allocating functions between humans and machines, but which, when generalized, relate to all functional allocation decisions. Principles 2 and 4 emphasize the creative nature of design that was emphasized in Chapter 8 on physical architectures; this creativity applies equally to the functional architecture and the allocated architecture. Principle 3 supports the use of decision analysis (see Chapter 13) for systematizing the decision process.

Capturing requirements for the refinement phase of the system's life cycle is the point of principle 5. The Vee model of the systems engineering process is compatible with principle 7. The process model for the allocated architecture, shown in Figure 9.1, supports principle 9.

TABLE 9.3 Price's Functional Allocation Principles

1. Allocation is part of design — allocation is one part of a larger process.
2. Allocation is invention — there is no formula for allocation, imagination is crucial to the success of the process.
3. Allocation can be systematized — the inclusion of imagination and invention does not preclude formalizing allocation as a rational decision process, combining invention and systematization yields a superior result.
4. Make use of analogous technologies building upon allocation decisions and their resulting successes and failures expands our allocation expertise.
5. Consider future technology — allocation decisions cannot be based on what exists now, but must address expected advances of technology.
6. Consider human optimization (realistic system implementation)—allocation cannot be based upon idealistic expectations of how the system will be realized, but should be based upon the likely capabilities of the system in its environment.
7. Use cycles of hypothesis and test — like any other part of system design, we are not smart enough to do it right the first time, so build in stages of and time for iteration.
8. Provide interaction — there are three design decisions that cannot be completely separated. The engineering decision of what the physical resources of the system are, the functional allocation of which functions will be performed by each system resource, and the detailed design decision that implements the allocation. There must be interaction amongst these decisions during the design process.
9. Provide iteration and decomposition — do not make the allocation final too quickly.
10. Develop tools of cognitive analysis. (human – machine allocation only).
11. Assure interdisciplinary communication — involve experts from all relevant fields in the allocation process.

The essence of Price's principles is that the allocation of functions to elements of the physical architecture involves conflicting objectives. Making this selection even more difficult is the fact that the systems engineering team has to evaluate objectives in more than one time span, for example, short-term performance versus future performance after possible upgrades have been completed. For these types of allocation decisions the decision analysis approach covered in Chapter 13 is recommended. The core of this approach is the use of an appropriate part of the objectives hierarchy that contains all of the key performance requirements and their stakeholder trade offs. Figure 9.4 illustrates such an objectives hierarchy for a hypothetical decision.

Another perspective on this allocation problem involves the use of design structure matrices. See Browning [2001] for more information design structure matrices. The design structure matrix (DSM) is meant to capture interactions of all sorts between functions so that intelligent combinations of functions into components can be derived. This is a bottom-up approach to the allocation problem, while we have previously been talking about this task as if it could only be approached from a top-down perspective. As discussed in the functional architecture chapter, there are many systems engineers who prefer the bottom-up approach.

As an example of a DSM application consider the creation of a development system architecture for the small block V-8 engine at General Motors [Eppinger, 1997]. This engine effort called 90% of the parts to be redesigned and 80% of the manufacturing equipment to be redesigned. As a result 22 product development teams (PDTs) were created, as shown in Figure 9.5. In an effort to determine the best way to organize the concurrent efforts of these PDTs, the interactions among the teams was documented and categorized as monthly, weekly, or daily. The matrix in Figure 9.5 is an example of a DSM. The three sized dots represent these three levels of interaction. Note the DSM is not symmetric because the rows represent where the input to a team are coming from while the columns represent which teams are receiving a given team's outputs. So the second column of the first row indicates which kind of interaction is needed for an input to the DPT A from DPT B. This is the opposite representation of an N2 diagram.

The main analytic concept behind DSMs is that the information in the matrix provides a clue as to how to rearrange the rows and columns so that clusters form along the diagonal of the reorganized matrix. These algorithms date back to the 1970s. Figure 9.6 shows such a rearranged matrix with four clusters along the diagonal for four aggregations of the DPTs that should prove very useful. Note the last DPT is the assembly DPT; it interacts with so many DPTs that it does not belong to any aggregate team.

So far the functional allocation decision process has been addressed as if the decisions had to be made during the design process and could only be modified during system upgrades. However, the computational resources that are now available for insertion into systems permit the design to include the real-time reallocation of functions to predefined resources. Typically this reallocation is

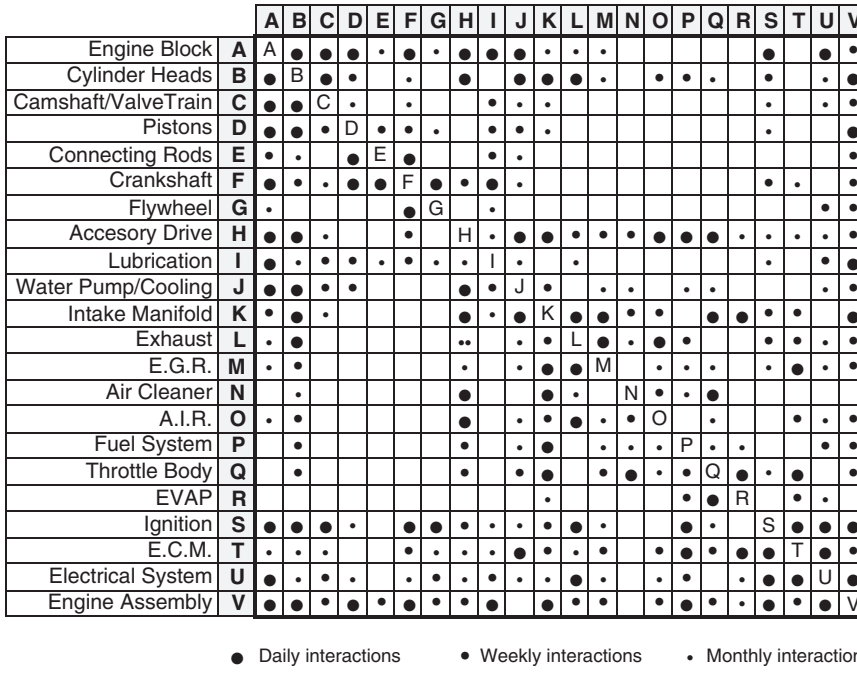

FIGURE 9.5 Interactions among PDTs for the small V-8 Engine Project at General Motors (after Eppinger [1997]).

between human and computer (hardware and software), or between one hardware resource and another, each running the same set of software. Examples of this dynamic reallocation include distributed processing architectures, parallel processing architectures, flexible manufacturing systems, and sophisticated command and control systems. This material is beyond the scope of this book; the interested reader is referred to Chu and Tan [1987], Gobinath and Gupta [1990], Levis et al. [1994], and Perdu and Levis [1993]. Jackson [2007] makes a strong case for an adaptive allocation of functions to components in order to develop more adaptive and resilient systems.

9.3.3 Finishing the Allocation Problem

Part of the critical documentation that is part of systems engineering is capturing the allocation of functions to the system and the system's components. Every bottom-level function in the functional decomposition should be allocated to one component of the physical architecture, or physical decomposition, as discussed in Figure 9.3. This physical decomposition begins with the system as the root of the tree. The top-level system function, or root of the

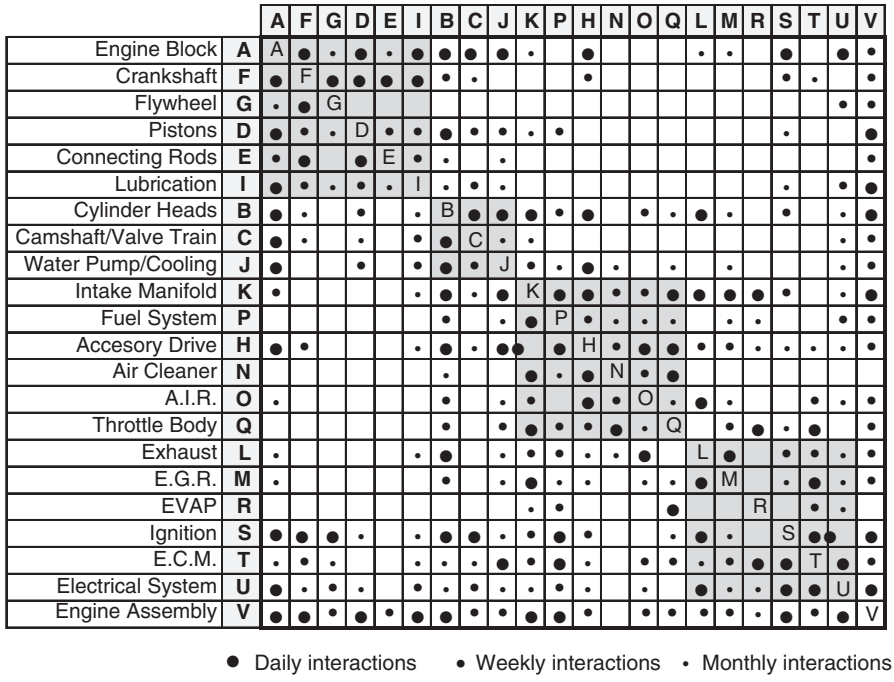

FIGURE 9.6 Reorganized DSM with four Aggregate teams (after Eppinger, 1997).

functional decomposition, is allocated to the system. The functions at the first level of functional decomposition are then allocated to one component on the first level of the physical decomposition. This allocation of the first level of functions may be the level of detail achieved in the first iteration through the engineering of the system (or first peel of the onion). In IDEF0 this allocation of functions to components is shown by adding the components as mechanisms to the functional architecture, thus creating a representation of the allocated architecture. See Figure 9.7 for an example of this depiction using IDEF0 (and the IDEF0 model in the elevator case study that can be downloaded from http://www.vitechcorp.com; see the section called allocated architecture). CORE utilizes an entity–relationship diagram (see Chapter 12) to show the allocation of functions to the system and the system's components. (CORE's System Description Document for the elevator case study shows the results of this allocation process.) Each iteration through the engineering of the system process adds another layer of bottom-level functions and components to the functional and physical architectures, respectively. Each bottom-level function will then be allocated to one component.

To obtain an executable model of the allocated architecture, later discussions will make it clear that the only allocation of functions to components that matters is the allocation of functions at the bottom of the functional

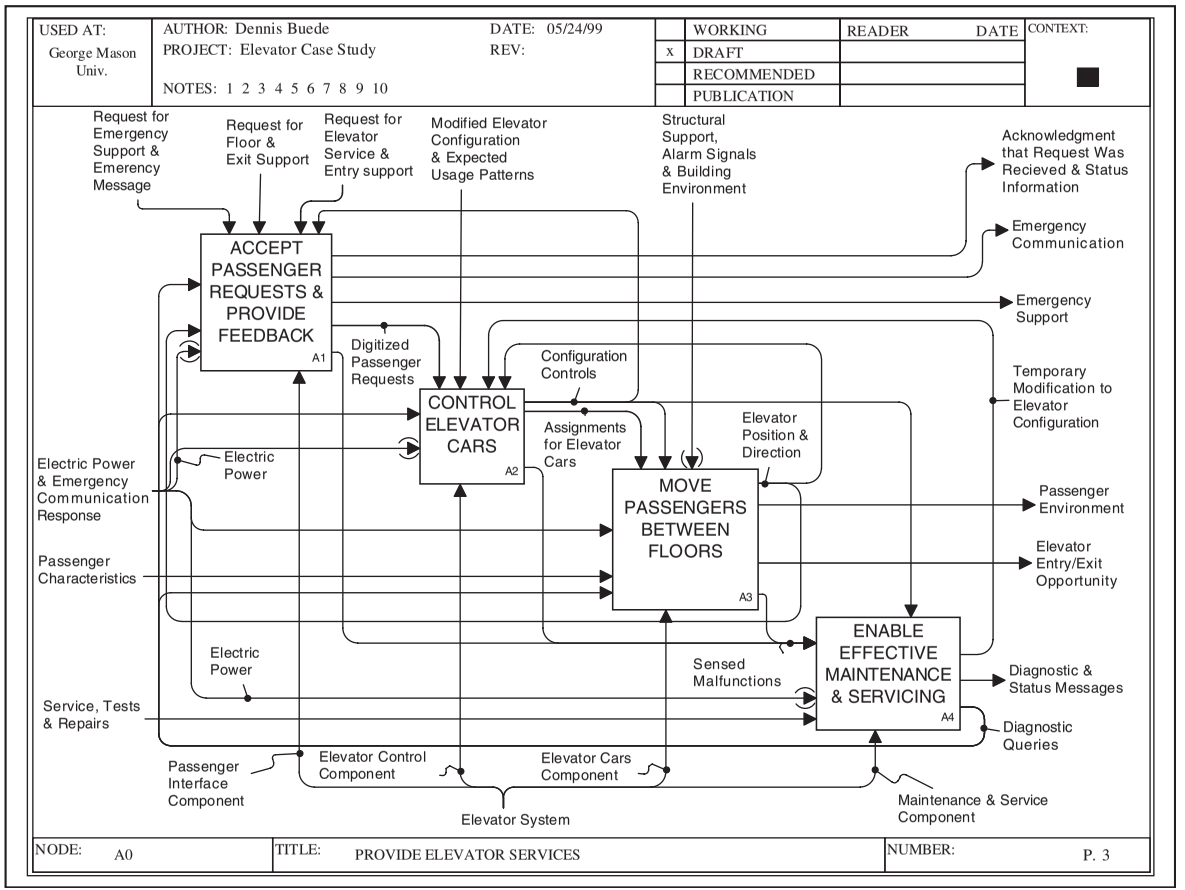

FIGURE 9.7 Allocating functions to components using IDEF0.

architecture to components at the bottom of the physical architecture. However, it is highly recommended that an executable model be created of the allocated architecture at several stages in the engineering of the system. Therefore, it is highly valuable to have a running record of the allocation of functions to components, so that this executable model is available at any level of abstraction needed.

As discussed in Chapters 6 and 7, there are tremendous benefits obtained by having the functional decomposition match the physical decomposition on a one-to-one basis. That is, for each function in the first level of the functional decomposition, there is one and only one component to which to allocate the function. In addition, every component must be allocated to one and only one function. This one-to-one mapping of functions to components must continue to the second and all subsequent levels of both the functional and physical architectures. (Note this definition of a one-to-one allocation of functions to components is consistent with the definition of a one-to-one function in Chapter 4.) Such a convenient mapping of functions and components can only occur if the functional and physical architectures are developed in concert with each other. The benefit of this one-to-one mapping is the ease with which input and output items can be allocated to external and internal interfaces. The true value of this matching will be covered in the next chapter.

9.4 TRACE NON-INPUT/OUTPUT REQUIREMENTS AND DERIVE REQUIREMENTS

In Chapter 7 on the functional architecture, the discussion of tracing requirements addressed the input/output requirements. These input/output requirements were traced to specific functions in the functional architecture. When the functions were allocated to the components as described above, these input/output requirements were associated with components. There remain several issues though to complete the derivation of requirements for each component in the allocated architecture: deriving additional input/output requirements for each function based upon internal items that the architecture needs, tracing system-wide and technology requirements to the system and deriving appropriate component-wide and technology requirements for each of the components, tracing trade-off requirements to the system and deriving trade-off requirements that are appropriate for each component, and tracing test requirements to the system, followed by the derivation of test requirements for each component.

9.4.1 Derive Internal Input/Output Requirements

Deriving input/output requirements based internal items that the system must create and use is not a difficult process if a graphical model (e.g., IDEF0, data flow diagram, or N^2 chart) of the functional and allocated architectures exists.

Once the functions have been allocated to the components, derived input/output requirements can be created based upon internal items (inputs and outputs) appearing in the functional architecture. Figure 9.7 shows the allocated architecture for the elevator case study that can be downloaded. There are five internal items that are created by one function and consumed by another function at this first level of the allocated architecture: digitized passenger requests, assignments for elevator cars, elevator position and direction, sensed malfunctions, and temporary modification to elevator configuration. A derived input and output requirement would have to be created for each of these items. Each of these derived input and output requirements would be traced to both the item and the functions responsible for consuming and creating the item, respectively. For example, Figure 9.7 shows that "Digitized Passenger Requests" is an internal item produced by the first top-level subfunction and sent to the second top-level subfunction. For this one internal item two derived requirements would be created:

The elevator system shall produce digitized passenger requests.
The elevator system shall consume digitized passenger requests.

Each of these derived requirements would be traced to the item "Digitized Passenger Requests"; the first derived requirement would be traced to the function "Accept Passenger Requests & Provide Feedback" while the second derived requirement would be traced to the function "Control Elevator Cars." Additional performance requirements for "Digitized Passenger Requests" would be created if appropriate.

9.4.2 Trace System-Wide Requirements and Derive Subsystem-Wide Requirements

Tracing the system-wide and technology requirements to the system is a very easy process. Almost all of these requirements will be traced to the system; although it is possible that some of these requirements should be traced to specific components that comprise the system. The most common example of this is a technology requirement such as "the system shall employ 'abc' technology." A technology requirement that can be traced to a subset of the components of the system should be.

However, the difficult portion of this task is the derivation of new requirements for the components based upon the system-wide requirements traced to the system. For example, there may be a cost requirement that says, "The system shall cost $1000 or less to use per month during its operation." How do we allocate, or "flowdown," this requirement among the components of the system?

Grady [1993] identifies three techniques that are used for flowdown: apportionment, equivalence, and synthesis. Apportionment spreads a system-level requirement among the system's components of the system, maintaining

the same units. Apportionment is appropriate for cost requirements; the system-level cost requirement is divided or apportioned out to the system's components, not necessarily in equal increments. Keeping a margin, 5 to 10%, in reserve as a risk mitigation strategy is not uncommon. For example, if the operating cost for the system is to be $1000 or less as suggested above for the elevator, the four components of the elevator shown in Figure 9.5 may be apportioned operating cost requirements of $40, $60, $800, and $50, respectively, with $50 held as risk mitigation.

Other examples for which apportionment is used are reliability, availability, and durability. In fact, the suitability (or quality or "-ilities") requirements are commonly apportioned from the system to the components. Note that it is not required that the apportioned values sum to the system-level requirement, as is the case of cost when the margin is included. If the system's components work in series, the component values for reliability will be larger than the system reliability. For the elevator case study the minimum threshold for reliability is 0.9, with a design goal of 0.99. The four components identified in Figure 9.5 all have to be operational for the elevator to be operational; so they are working in series. The apportioned reliability thresholds for these components may then be 0.96, 0.995, 0.96, and 0.99; the product of these four numbers is 0.91, which provides a margin of a bit less than 0.01 for risk mitigation. Similarly, there would be design goals apportioned to the four components of 0.996, 0.9995, 0.996, and 0.999, respectively. An example of a derived reliability requirement is:

> The elevator component, Passenger Interface, shall have a reliability of 0.96 or greater. The design goal is 0.996.

Equivalence is a simple flowdown technique that causes the component requirement to be the same as the system requirement. An example of a requirement to which equivalence is appropriate is "the system shall be olive green in color." Requirements for which equivalence is appropriate for flowdown are almost always constraints.

The more complicated technique for flowdown is synthesis. *Synthesis* addresses those situations in which the system-level requirement is comprised of complex contributions from the components, causing the component requirements that are flowed down from the system to be based upon some analytic model. The system-level requirement will have significantly different units than the derived, component requirement has. In this case an analytic or simulation model must be developed and analyzed to determine how to take the system-wide requirement and derive component requirements. In fact, this approach is most often used to derive requirements associated with outputs or inputs of the system, such as accuracy, range, or thrust. For the elevator case study, there is an output requirement relating to the average time between the passenger making a request and being delivered to the requested floor. This system-level requirement would be flowed-down via synthesis to all four components shown in Figure 9.7.

9.4.3 Trace Trade-Off Requirements and Derive Subsystem Trade-Off Requirements

Deriving trade-off requirements that are appropriate for each subsystem follows tracing the system's trade-off requirements to the system. This derivation is based upon the system-wide trade-off requirements. This step is the third element of requirements derivation that is part of finishing the allocated architecture.

The trade-off requirements developed for the system all address trade offs for cost, for schedule and performance, and for cost with schedule and performance; tracing all of these requirements to the system is therefore appropriate. Each of these trade-off requirements is related to an individual input, output, or system-wide requirement. Based upon the derivation of requirements for each of these input/output or system-wide requirements, it is straightforward to develop an objectives hierarchy for each component, as shown in Figure 9.8. Generally every element of the system's objectives hierarchy that is related to a system-wide requirement will also become part of the objectives hierarchy for each component; cost, schedule, and suitability requirements are generally flowed down to every component, as discussed above. Similarly, it is inappropriate to create a component-wide requirement when there is no system-wide requirement from which the component-wide requirement can be derived.

Before moving on to input/output requirements, the derivation of ranges for each system-wide requirement, the associated value curve over the derived range, and the weight to be assigned to that range must also be addressed. First, the two extremes of the value range must be flowed down from the system to each component. This should have been done as part of the flowdown process described above.

The value curve assigned to this derived requirement should ideally have the same shape as that for the system-wide requirement. However, an example using reliability can be shown as a counterexample for successfully communicating a consistent value function from the trade-off requirements at the system level to the trade-off requirements across the components. Reliability is chosen here because the system's reliability is known to be a nonlinear function of the reliabilities of the components of the system. Suppose the value function for the system's reliability was defined by an exponential function exhibiting decreasing returns to scale. Decreasing returns to scale indicates that unit improvements in the reliability near the threshold of minimum acceptability would have much greater value to the stakeholders than unit improvements near the design goal. This concept of decreasing returns to scale is common in the economics and decision analysis literature; see Chapter 13 for more details. Suppose the minimum acceptable system reliability is 0.9 and the design goal is 0.99. There are two components acting in series that comprise the system. Each of these components is given a threshold of minimum acceptable reliability equal to the square root of 0.9 (or 0.95) and a design goal of the square root of 0.99 (or 0.995). The value curve for the system reliability and the reliability of each component is

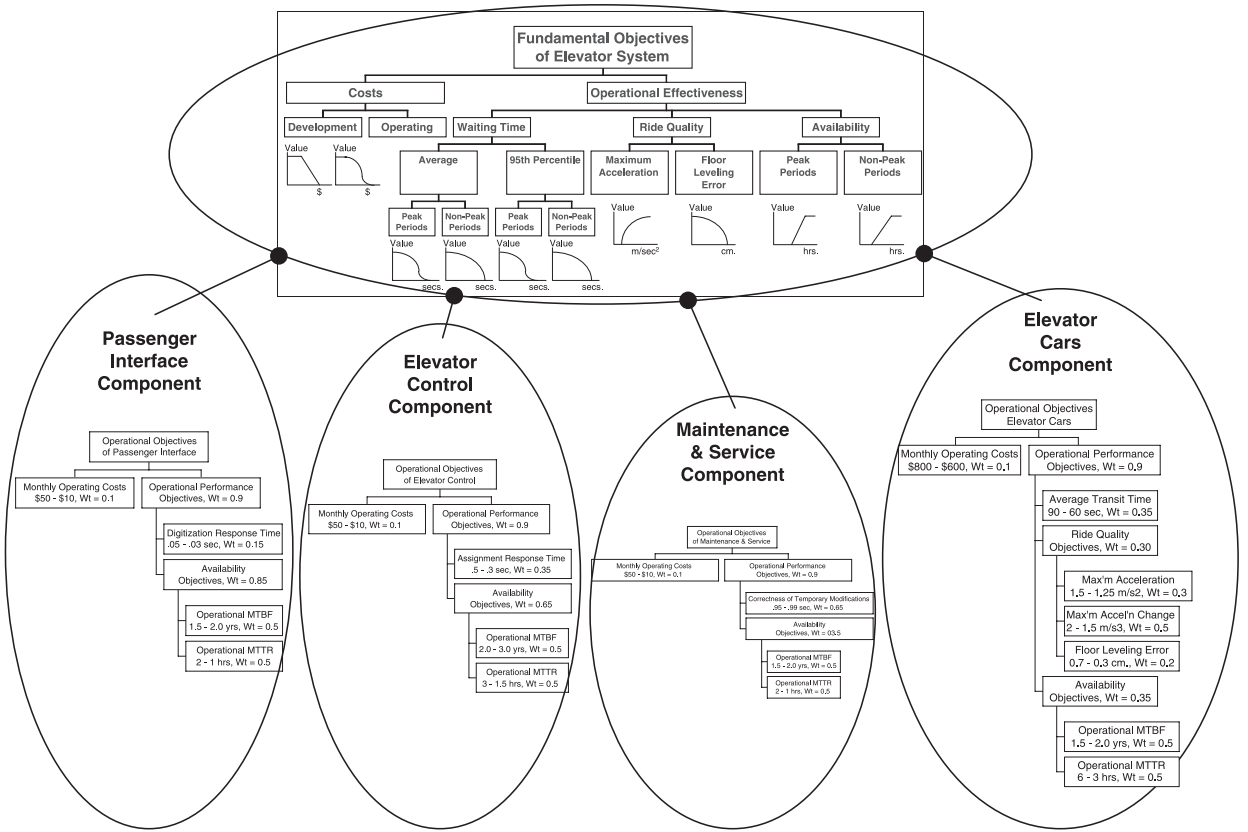

FIGURE 9.8 Derived objectives hierarchies for the elevator case study.

assumed to have the same form, $(1 - e^{-\alpha(r-r_{\min})})/(1 - e^{-\alpha(r_{\max}-r_{\min})})$, that the value curve for system reliability had. The parameter, α, determines the shape of the curve. When α equals 1.0, the curve is linear. The greater α is above 1.0, the greater the bow in the curve and the greater are the decreasing returns to scale. Figure 9.9 shows the value curve for system reliability on the left for values of α from 30 to 1.

The right-hand graphs in Figure 9.9 show the value for system reliability as a function of the reliability for the first component, X, when the system reliability is held constant at 0.9439. In each of the graphs on the right the value is computed by the weighted average of the values for the reliabilities of the two components:

$$\text{Value} = 0.5\,\text{v}(\text{reliability of component X}) + 0.5\,\text{v}(\text{reliability of component Y})$$

The weights for the two components are assumed equal since the distance from threshold to goal is the same for both. As can be seen in Figure 9.9, the value for the reliabilities of the two components is not constant over the range of values for the reliability of the first component, even though the reliability of the system is being held constant. The numbers to the far right of Figure 9.9 show the value for the system's reliability when the system's reliability is held constant at 0.9439; the values in the right-hand graphs are also not equal to these numbers except for the case of the linear value curves. This suggests that only linear value curves should be used for trade-off requirements.

The final issue in deriving trade-off requirements for each component concerns those trade-off requirements that address quality, quantity, or timeliness of the system's inputs or outputs. Each of these input and output requirements will already have been traced to a function that was allocated to a component. Therefore each trade-off requirement for an input or output can already be associated with one component, assuming the allocation mapping of the input/output requirement to functions was one-to-one. A complicating issue, however, is that there may be good reasons to create a trade-off requirement for an input or output requirement that was derived on the basis of the need for an internal item produced and consumed by the functional architecture. An example of this in Figure 9.7 is the "Digitized Passenger Requests" for the first sub-function. This internal item is related to the elevator objective of "Waiting Time" shown in Figure 9.8. Such a trade-off requirement must be traceable to a performance aspect of a stakeholders' input/output requirement; nonetheless, it is the only case when the objectives hierarchy will have an element that is not identical to an element of the system's objectives hierarchy.

9.4.4 Trace Qualification Requirements and Derive Subsystem Qualification Requirements

The final element of completing the requirements development for each individual component is tracing the qualification requirements to the system and then deriving qualification requirements for each component. Recall that

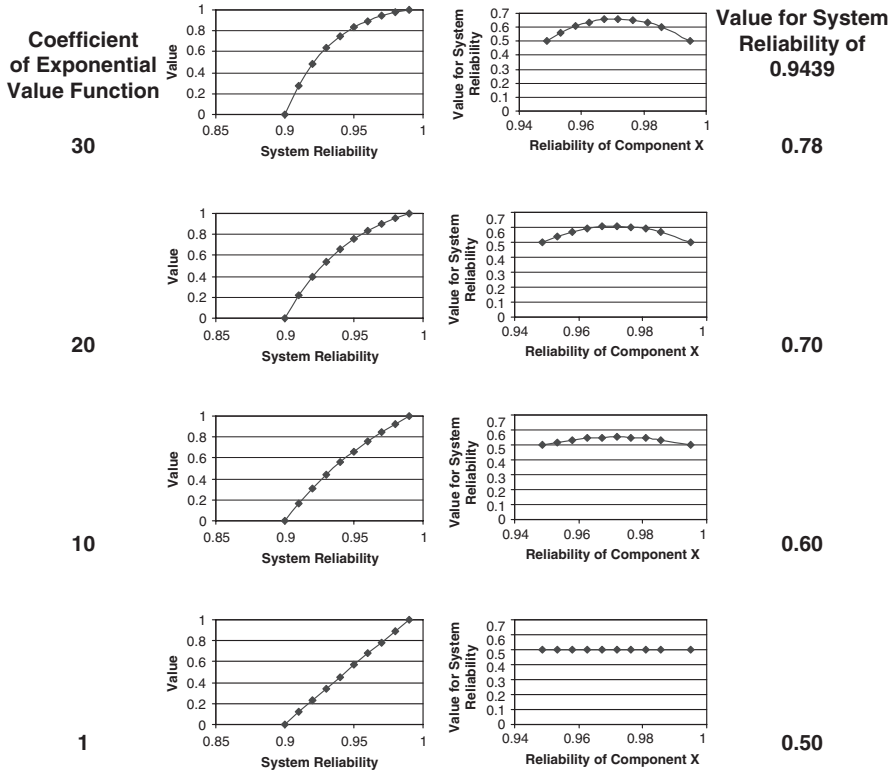

FIGURE 9.9 Sensitivity of value for system reliability trade offs to derived trade offs for component reliabilities.

the four categories of qualification requirements are observance, verification plan, validation plan, and acceptance plan. These last two categories only apply to the system. Therefore, after all qualification requirements have been traced to the system, derived requirements for the components are developed only from the first two categories (observance and verification). This derivation process is quite straightforward; observance requirements relate to specific input/output and system-wide and technology requirements. Therefore, deriving observance requirements follows the derivation process of input/output and system-wide and technology requirements. Deriving a verification plan for each component should be relatively straightforward, given the verification for the system.

9.5 DEFINE AND ANALYZE FUNCTIONAL ACTIVATION AND CONTROL STRUCTURE

When discussing IDEF0 (Chapter 3) and functional decomposition (Chapter 7) the need for activation and termination criteria was mentioned. That is, there

are criteria that need to be established for each function; these criteria determine what set of inputs (and associated values) will activate the function and what set of outputs (and associated values) are sufficient to terminate the function. The bottom-level functions in the functional architecture must have their activation and termination criteria completely specified. The intermediate- and top-level functions are aggregates of the bottom-level functions and as such are for modeling purposes only; intermediate- and top-level functions do not have or need activation and termination criteria. However, recall the previous discussions of peeling the onion and the fact that the bottom-level functions of an early peel of the onion become intermediate-level functions during later peels of the onion.

In addition to the activation and termination of a function, the conditions under which one function precedes or follows another function's processing must be clearly defined. Examples of approaches to defining such precedence conditions can be found in Chapter 12 under behavioral modeling. Most of these behavioral modeling methods allow the dynamics of the system to be explored by providing an executable model of the system's functions. These executable models are either discrete-time or discrete-event simulations when implemented on a computer. The reader is referred to Chapter 12 for more detailed discussions on this subject.

Before discussing the dynamic issues associated with the performance of a system, the balancing or aligning [Yourdon, 1989; Schmekel and Wingard, 1993] of multiple models of a system should be addressed. At this point the functional architecture contains a data model and a process model of the system in question. The generation of activation and termination conditions for each function plus the control structure associated with the concurrent or asynchronous behavior of functions with respect to each other is contained in the behavioral model of the system. Yet each of these models contains overlapping data elements: Inputs and outputs are in all three models, and functions are in the process and behavior models. These models better be consistent and coherent representations of each other or their results will be worthless to the engineers of the system; in essence, the engineers will have modeled several different systems while thinking they were addressing only one system. Schmekel and Wingard [1993] present the most complete treatment of this topic known to the author.

There are several benefits of executable models. First, the design can be explored to find major design flaws that are manifested as deadlocks, livelocks, starvation, surge or race conditions, or oscillatory conditions. The second major benefit is to permit the systems engineering team to assess the degree to which the design meets various timing and throughput requirements.

Deadlock, livelock, starvation, surge (race), and oscillation are dynamic characteristics that are not desired in dynamic, time-varying systems. *Deadlock* is an undesired state of the system in which activity ceases and throughput is nonexistent. Deadlock can occur for two reasons: contention over resources and waiting for a communication [Levi and Agrawala, 1994]. Contention over

resources occurs when each of several components requires the same resource for a task, but none of the components is willing to free the resources it has accumulated. As a result activity stops while the components wait for additional resources to complete their assigned tasks. Waiting for a communication occurs when various components are attempting to synchronize their actions or verify their status; in either case each component enters a state called "wait for communication," but the communication never arrives because the components are in a strongly connected wait state.

Deadlock associated with resources is often described using the "dining philosophers" problem. There are five philosophers sitting around a circular table preparing to eat spaghetti. There are five forks, one between each of the adjacent philosophers. Before eating the spaghetti each philosopher requires two forks to move the spaghetti from the bowl in the middle of the table onto her/his plate. If each philosopher grabs (and locks) the fork on the left, no philosopher will be able to eat; this is deadlock. The solution requires the creation of a conditional locking mechanism on the forks by the philosophers that ensures that each philosopher obtains both forks for a limited time to move the spaghetti to her/his plate. After completing this initial task, each philosopher then releases both forks for a period of time. Once each philosopher has spaghetti on her/his plate, then only one resource is required by each and all five philosophers can eat simultaneously.

Graph theory is often used to depict the resource sharing problem with what is called a "wait-for-resource" graph. Define each component as a node. Define the relation R to be "awaits a resource possessed by." Figure 9.10 shows a system with four components in which there is a potential deadlock involving the first three components. Mathematically, it can be shown that any system having a wait-for resource graph with a cycle can become deadlocked if several other conditions apply [Levi and Agrawala, 1994]. If there are many components and the wait-for-resource graph is complex, the existence of a cycle may not be obvious by inspection. Typical solutions to eliminating or reducing the chance of deadlock due to resource contention are to oversize buffers and resource pools, reduce the concurrency of operations, add delays, institute a manual or automated deadlock detection and recovery process, and allow preemption of locked resources. Ferrarini and Maroni [1997] define three generic categories of options: avoidance, prevention, and recovery.

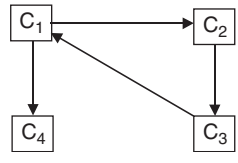

FIGURE 9.10 Wait-for-resource graph depicting deadlock.

A "wait-for-communication" graph can be used to examine the possibility of deadlock due to communication. In this case a cycle (with other conditions) is not sufficient to guarantee deadlock; a strongly connected, cyclic graph is necessary. Deadlocks have been studied in communication systems [Duato et al., 1997] for a long time and procedures have been embedded into most communications protocols to break communication deadlocks when they occur.

Livelock is a dynamic condition with the same result as deadlock but for a different reason. In deadlock the system (or part of the system) halts activity because various activities are holding or utilizing resources needed by other activities. In *livelock* the resources are being routed in cycles (oscillating) while waiting for the proper allocation of resources to enable the completion of necessary activities; unfortunately this proper allocation of resources is never achieved and the system cycles continuously, never reaching the desired outputs. In communication networks livelock can only occur when information packets are permitted to traverse paths that are not minimal.

Starvation occurs when a function needs a particular resource for execution, but the resource is always allocated to other functions due to a poorly designed resource assignment algorithm. This condition is one that can be found with little trouble as long as a reasonable effort is made to model the dynamics of the system. However, it can easily be overlooked if no effort is devoted to examine the system's dynamic properties.

The dynamic condition called *surge* or *race* occurs in relatively uncontrolled systems when components are competing with each other to perform a task. A common example is found in older elevator systems during nonpeak times; a potential passenger pushes the up button and *observes* that all of the stationary elevator cars are converging on her floor. She gets into one of the elevator cars. The next passenger now pushes the down button and the remaining elevator cars surge to that passenger. The surge condition is a waste of resources while it is occurring and can leave the system in an undesirable state for future tasks; all of the elevator cars but one will end up waiting at the same floor for future passengers.

These negative dynamic conditions can be designed into a system inadvertently without the engineers' knowledge *unless* the designers undertake a detailed study of their design. Discrete-event simulations involving Petri nets, queueing theory, behavior diagrams, or extended function flow block diagrams are needed to investigate the design of the system via mathematics and simulation and to understand the degree and extent of such negative behaviors. Naturally, if negative behaviors exist, design changes can be examined to eliminate or minimize them.

9.6 CONDUCT PERFORMANCE AND RISK ANALYSES

A wide range of quantitative analyses is commonly performed during the system development process that fits within the categories of performance,

trade-off, and risk analyses. The parametric diagrams of SysML can be used to design and document these analyses. In fact, these analyses can be considered a system in their own right.

Risk analyses are often completed at the beginning of the development process to examine the major design options under consideration. For example, at the earliest stage of development the systems engineering team should consider a range of divergent concepts. A *risk analysis* examines the ability of the divergent concepts to perform up to the needed level of performance across a wide range of operational scenarios. At this time there remains substantial uncertainty about the stakeholders' needs, the state of technology under consideration, and the details of the allocated architecture. The relative costs and schedule implications of the various concepts also have to be taken into account. This is where the stakeholders have to debate how much money and time they are willing to pay for increased performance in selected operational scenarios. Addressing uncertainty and multiple objectives in these early risk analyses is critical; see Chapter 13.

Performance analyses are for the purpose of discovering the range of performance that can be expected from a specific design or a set of designs that are quite similar. The performance parameter in question can be associated with an output of the system or with a system-wide metric; in either case there is almost always a related objective in the objectives hierarchy and an associated performance requirement. These performance analyses usually take the shape of engineering models and simulation models. The simulation models may be deterministic or stochastic, depending on the issue involved and experience level of the design team with the technology.

Common system-wide performance analyses address operational feasibility issues such as reliability, availability, maintainability, usability, supportability, durability, and affordability. Similarly, performance analyses are conducted to address concurrent engineering issues related to the impact of the operational system design on the manufacturing, deployment, training, and disposal systems. Blanchard and Fabrycky [1998] provide detailed discussions of many of these topics: design for reliability, for maintainability, for usability, for supportability, for producibility and disposability, and for affordability. References for detailed analysis of cost, reliability, maintainability, and availability include Blanchard and Fabrycky [1998], Frankel [1988], Pages and Gondran [1986], Pohl [2007], Pohl and Nachtmann [2007], and Sage [1992].

Some organizations have dictated that the system be designed to cost; that is, there is a cost constraint, and the engineering design team has to guarantee that the system will meet this cost constraint. Design-to-cost works best by designing a reduced-capability system with various optional features that can be added if the cost estimates are low.

A *trade study* focuses on finding ways to improve the system's performance on some highly important objective while maintaining the system's capability in other objectives. Trade studies are focused on comparing a range of design options from the perspective of the objectives associated with the system's

performance and cost. For example, aircraft manufacturers always do trade studies focused on the aircraft's weight, while maintaining the system's cost, safety, and so forth. Similarly, safety, reliability, and cost are among the many other objectives that are commonly the focus of a trade study.

9.7 DOCUMENT ARCHITECTURES AND OBTAIN APPROVAL

Documenting the system design completely is important. Not only should the key elements of the requirements process (operational concept, external systems diagram, objectives hierarchy, and requirements), and the three architectures (functional, physical, and allocated) be documented, but also the audit trail for how the results were obtained and why they are what they are. In every system development activity there are many occasions during the life of the system when engineers will want to find out why a particular part of the design is the way it is. This curiosity usually arises because the engineers want to change the design and need to understand the original rationale for the current configuration; there may have been some issues that the current engineers have not thought of that would keep them from making the change they are contemplating. Unfortunately, it is rare to talk to an engineer who went looking for design rationale on any type of a system and was successful. The design decisions that are made intuitively and on the spur of the moment (often without even realizing that a key decision is being made) are seldom documented. The design decisions that are made consciously with an explicit analytical approach, such as decision analysis (see Chapter 13), will be very well documented as long as the analysis material is archived properly.

Obtaining approval of the system's design, or allocated architecture, typically requires long meetings with many members of the engineering team and representatives of the stakeholders. A number of key design decisions are revisited, arguing for the value of the systematic development and archiving of the rationale for these decisions. Once the system's allocated architecture is approved, it is quite simple to develop a specification for each subsystem with the information that is available.

9.8 DOCUMENT SUBSYSTEM SPECIFICATIONS

At this point the system design is complete and each major subsystem of the system can be documented in terms of its own operational concept, external component diagram, objectives hierarchy, and requirements document. The requirements document for each component, commonly called a specification (or spec for short) includes input/output, technology and subsystem-wide, trade-off, and qualification requirements.

Shortly after the subsystem design activities are initiated, a preliminary design review should be held with the stakeholders to obtain their input and approval for proceeding further with the subsystem design.

9.9 SUMMARY

The allocated architecture combines the physical and functional architectures so as to meet the stakeholders' requirements and related derived requirements. This combination of the physical and functional architectures requires the allocation of functions to physical resources; at this point the system's design can be simulated and analyzed in terms of the stakeholders' requirements and operational concept of the stakeholders. As the physical and functional architectures are integrated, the interfaces of the system (both external and internal) can also be defined and designed.

The processes that comprise the development of the allocated architecture are the allocation of functions to components, the tracing of system-wide requirements to the system, the derivation of requirements, the definition and analysis of functional activation and control structures, the conduct of performance and risk analyses, documenting the allocated architecture, and documenting the specifications.

The allocation of functions to physical resources was addressed in terms of the appropriate objectives for this major decision. From a historical perspective the most difficult allocation decision is machine versus human. The allocation between hardware and software is also discussed. Ultimately, this allocation process requires trade offs between fast and accurate performance of tasks versus ability to upgrade and change the processes for performing the tasks. As such, decision analysis (see Chapter 13) should be used to evaluate alternate allocation options in terms of the objectives of the stakeholders.

To complete the component specifications additional requirements (input/output, system-wide and technology, trade-off, and qualification) must be derived from those that are already available. Examples of these derivations are provided. Three methods for flowing down requirements that were initially traced to the system are also described.

Critical system-wide issues associated with functional activation and control are discussed here. These issues include deadlock, livelock, starvation, and surge (or racing) of the system.

Decision analysis is discussed as a normative model for conducting risk analyses, performance analyses, and trade studies. An illustration of a risk analysis was provided.

The design process has been likened to peeling an onion throughout this book. The development of the allocated architecture should proceed as though an onion were being peeled. The first allocated architecture developed should be for the subsystems of the system at a high level of abstraction (low level of detail). Then the entire process is repeated at a lower level of abstraction (greater detail) for the components of the subsystems, consistent with the Vee model discussed in Chapter 1. This repetition at lower and lower levels of abstraction yields allocated architectures at higher and higher levels of detail. The advantage of this approach is that as each new peeling begins the engineers for each component can work their design processes in relative seclusion from

the engineers for other components. Each group of engineers has interfaces between their components and other components and the external systems that have been defined at an appropriate level of detail, yielding a coherent set of requirements with which to work. The work of these several teams of engineers will need to be integrated and coordinated at the newest level of detail before the allocated architecture can be complete for this more detailed level of abstraction.

CASE STUDY: WIDE AREA AUGMENTATION SYSTEM OF THE FEDERAL AVIATION ADMINISTRATION (FAA)*
* PROVIDED BY TIM PARKER

The objective of the U.S. FAA Wide-Area Augmentation System (WAAS) is to provide a navigation aid, for use by commercial and general aviation that is derived from the global positioning system (GPS) standard positioning service (SPS). (GPS employs a constellation of 24 satellites, each of which continually broadcasts its position at the time of broadcast.) The GPS satellites provide the radio frequency equivalent of a navigator's optical star fix. However, the accuracy and integrity of the SPS broadcast is not the ultra-high quality that the FAA requires to ensure the safety of civilian aircraft passengers and operators. Therefore WAAS determines the position of the GPS satellites more precisely than the SPS, and broadcasts "corrections" in real-time.

To validate the competing designs the FAA required each bidder to develop a special analysis tool known as a service volume model. The goal being that specific aspects of the performance of a given system design could be easily synthesized and simulated using computers. The results of the simulations are then useful for understanding the effects of flowing down certain performance allocations as requirements on lower tier system components such as the placement and number of ground monitoring antennas used for observing the GPS satellites. Because the simulation is capable of representing the dynamic nature of the spacecraft orbits, the tool can analyze the effects of outages resulting from individual or combinations of component failures (i.e., satellites and antenna monitor sites). In the case of this particular procurement the FAA included a task in the statement of work that described the use of the simulation tool for determining the exact number and location of the monitoring antennas.

The top-level requirements, that the WAAS simulation helps to explore, are the selection and geographic location of the ground monitor sites used to observe the GPS satellites, the number and location of geostationary satellites used to broadcast the corrections, and the coverage area or service volume where the WAAS service is available for use.

Additionally, the simulation accounts for certain a priori aspects of the models used to represent the effect on system performance from such phenomena as pseudo-range measurement error due to receiver noise, signal propagation delay due to the ionosphere, satellite clock estimation error, and satellite clock dither prediction error, that is, selective availability [Braasch, 1990; Kee et al., 1991]. The flowdown to the components is quite involved since there is a dynamic relationship among the number and geographic location of the monitor sites, the UPS satellites and the a priori characteristics of the systems algorithm. Suffice it to say that an acceptable result based upon specific a priori assumptions could flow to several components, the allowable receiver noise at the ground monitor site, the location and number of ground monitor sites (i.e., ground monitor site geometry), the number and location of the geostationary satellites for broadcasting the corrections, and the resulting coverage area or expected service volume, which is a function of both the geosatellite antenna pattern (i.e., foot print) on the surface of Earth as well as the geometry of the ground monitor sites.

To support the precision approach phase of aircraft flight operations, WAAS must deliver data to the user in the form of corrections for each UPS satellite's position and clock. This data, when applied to determine the position of a given user, should yield an answer that is accurate to better than 7.6 meters (in both the vertical and horizontal dimensions) 99.9% of the time throughout the coverage area.

A simple way to recognize how this relates to the problem of determining the number and placement of the monitor sites is to first understand that the problem that WAAS solves is essentially the navigation satellite user's problem inverted. By this we mean that normally the user of the GPS is concerned with tracking at least four satellites whose spatial relationship to each other and to the user, represented by a unit less value known as geometric dilution of precision (GDOP), satisfies the expression GDOP < 7. Visualize this relationship as an inverted pyramid with the user at the apex and each of the four vertices of the base representing a GPS satellite. Simultaneously solving the equations for the range measurement between the user and each of the observed UPS satellites yields the user's position.

Now recall that the problem that WAAS must solve is to correct the broadcast position and clock of each observed GPS satellite based upon the precisely known location of a set of ground monitor stations. Imagine the ground monitor sites as independent observers of the UPS satellites sharing a universal clock. For a given satellite's position, the ground monitor stations become the vertices of the base of a polyhedron whose vertex is represented by an observed GPS satellite. The spatial relation between the monitor stations and the satellites is analogous to the relation between the user and the satellites. Through the use of a continuous Kalman filter the WAAS arrives at an ensemble solution

for each satellite that is observed by its network of ground monitor stations.

The top-level physical architecture for WAAS allows for up to 70 monitor sites to be constructed and networked into four master control sites. As might be expected with any complex system of this nature, the nonrecurring engineering costs are daunting and every effort is made to reduce them. Naturally the FAA would not build all 70 ground sites and then determine if fewer could be used. Instead the simulation tool is utilized to predict the system performance when specific combinations of components are synthesized together as a working system. Early results published by Lockheed Martin Federal Systems (LMFS) (prior to acquisition by Lockheed, LMFS was originally the Federal Systems Division of IBM) indicated that based upon their simulation results a far smaller number of ground antennas would be necessary. The analysis used the LMFS Service Volume Model (SVM), a high-fidelity covariance-based simulation tool used to determine user obtainable navigation accuracy and service availability.

In addition to these analysis results LMFS undertook the development and fielding of a Wide-Area Differential Global Navigation Satellite System (GNSS) Testbed; see Figure 9.11 for the physical architecture block diagram. The purpose of the testbed, like the simulation, was to further develop knowledge about the allocated architecture and confirm the performance of the algorithms being considered for use on WAAS. A critical activity during the testbed's life cycle was its deployment into an operational environment. For this task the SVM simulation tool was used to determine optimal locations for the GPS receivers and ground antennas.

The top-level system objectives to be optimized for the testbed are easily expressed as: (1) minimize user range error, (2) maximize the area of geographic coverage where the user range error is 7.6 meters or less, 99.9% of the time, and (3) minimize the cost (i.e., deployment and operational). The first two components require the use of the simulation while the third component is treated as a simple linear projection of the costs incurred from acquiring the testbed equipment, leasing test laboratory space, and paying periodic operational expenses (i.e., telephone, electrical, technical personnel, and miscellaneous). The results of the simulation were combined with cost data for the prospective sites and evaluated using a simple multiattribute value analysis technique, which considered the top-level system objectives. Note that the deployment costs were determined to be roughly equal and for purposes of the analysis were considered to be equal among each set.

Many preliminary studies were undertaken to identify candidate locations for the ground monitor sites. Typically these were in the eastern half of the United States and within close distances to one another to minimize travel time for deployment and maintenance.

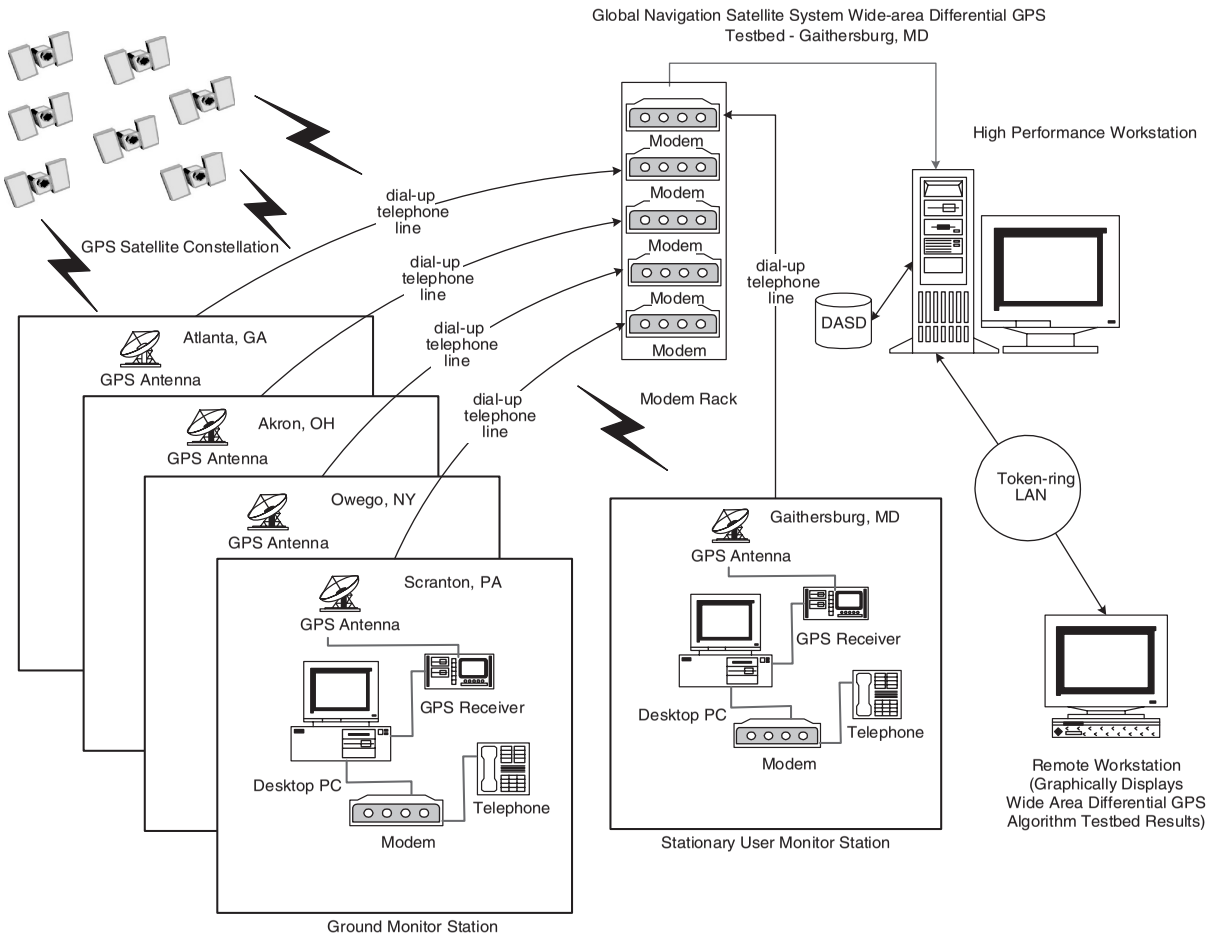

FIGURE 9.11 Physical architecture diagram for GNSS testbed.

Of the many possible sites, several were conveniently collocated with an existing company facility. The site combinations were evaluated together, as a location set. Different sets, or combinations of the sites, were evaluated using the SVM to determine if there would be a significant effect on the expected GNSS testbed performance. The multiattribute value analysis of the combined simulation and cost results is summarized in Table 9.4. The table contains error values representing the SVM prediction for average vertical position accuracy (VPA) and average horizontal position accuracy (HPA) as well as the average user error, where user error is defined as the root-sum-square of VPA and HPA. Coverage represents the percent of evaluated grid points where the predicted accuracy at each point is less than the required threshold value of 7.6 meters.

Coverage for sets 2 and 3 were significantly worse than sets 1 and 4, providing justification to eliminate sets 2 and 3 from consideration. Although set 4 meets the objectives of maximum coverage and minimum user error, the high operational cost of set 4 due to the usage of non-company property makes set 4 look inferior to set 1. Set 1 was preferred because it offered a reasonable geometry for determining wide-area corrections, had good coverage, and offered a smaller operational cost even though the average user error was the worst. The average user errors were all so close to each other that this objective was not very meaningful in discriminating among the alternatives.

TABLE 9.4 SVM Site Location Analysis Summary

Set	VPAA	HPA	User Error	Coverage	Monthly Operational Cost	Sites
1	7.013	7.358	5.082	84%	100	O, G, Ak, At
2	6.871	7.219	4.983	36%	125	O, G, Ak, N
3	6.837	7.187	4.960	36%	105	O, G, Ak, S
4	6.829	7.1	4.953	84%	130	O, N, Ak, S

Site Key	*Location*
Ak	Akron, OH
At	Atlanta, GA
G	Gaithersburg, MD
N	Norfolk, VA
O	Owego, NY
S	Scranton, PA

PROBLEMS

9.1 For the ATM system:

i. Allocate your functions to one or more of ATM's components.
ii. Trace your system-wide and technology requirements to the ATM system or one or more of its components.
iii. Derive component-wide requirements for each system-wide requirement and allocate the appropriate derived requirements to your components.
iv. Print a System Description Document for ATM.

9.2 For the OnStar system:

i. Allocate your functions to one or more of OnStar's components.
ii. Trace your system-wide and technology requirements to the OnStar system or one or more of its components.
iii. Derive component-wide requirements for each system-wide requirement and allocate the appropriate derived requirements to your components.
iv. Print a System Description Document for OnStar.

9.3 For the development system for an air bag system:

i. Allocate your functions to one or more of the development system's components.
ii. Trace your system-wide and technology requirements to the development system or one or more of its components.
iii. Derive component-wide requirements for each system-wide requirement and allocate the appropriate derived requirements to your components.
iv. Print a System Description Document for the development system.

9.4 For the development system for an air bag system:

i. Allocate your functions to one or more of the manufacturing system's components.
ii. Trace your system-wide and technology requirements to the manufacturing system or one or more of its components.
iii. Derive component-wide requirements for each system-wide requirement and allocate the appropriate derived requirements to your components.
iv. Print a System Description Document for the manufacturing system.

9.5 A system that is available 90% of the time is said to have one "9" of availability. Of the 365 days in a year, such a system would be "down" about 36 days and 12 hours.

i. A system that is available 99% of the time has two "9's". How many days and hours per year is this system "down"?

ii. How many days, hours, and minutes is a system with three "9's" of availability down?
iii. How many hours, minutes, and seconds is a system with four "9's" of availability down?
iv. How many minutes and seconds is a system with five "9's" of availability down?
v. How many minutes and seconds is a system with six "9's" of availability down?
vi. Where does the general class of personal computers fall in this spectrum of availability? Where do you think the air control system of the Federal Aviation Administration for a country should fall in this spectrum? Where does the telephone system fall? Where does your Internet provider fall?