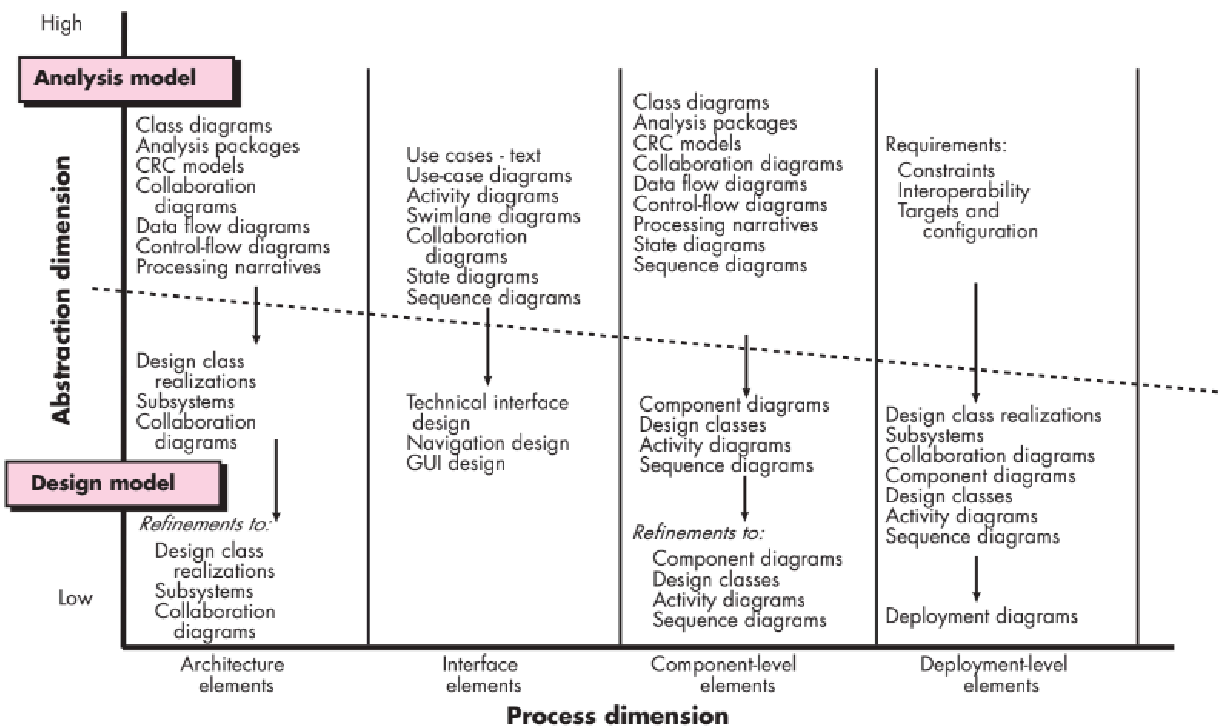
- The model of software is defined as a set of design classes.
- Every class describes the elements of problem domain and that focus on features of the problem which are user visible

❖ The design Model

The design model can be viewed in two different dimensions as illustrated in Figure 8.4. The process dimension indicates the evolution of the design model as design tasks are executed as part of the software process. The abstraction dimension represents the level of detail as each element of the analysis model is transformed into a design equivalent and then refined iteratively. Referring to Figure 8.4, the dashed line indicates the boundary between the analysis and design models. In some cases, a clear distinction between the analysis and design models is possible. In other cases, the analysis model slowly blends into the design and a clear distinction is less obvious.

The elements of the design model use many of the same UML diagrams7 that were used in the analysis model. The difference is that these diagrams are refined and elaborated as part of design; more implementation-specific detail is provided, and architectural structure and style, components that reside within the architecture, and interfaces between the components and with the outside world are all emphasized.

FIGURE 8.4 Dimensions of the design model



Data Design Elements

Like other software engineering activities, data design (sometimes referred to as data architecting) creates a model of data and/or information that is represented at a high level of abstraction (the customer/user's view of data). This data model is then refined into progressively more implementation-specific representations that can be processed by the computer-based system. In many software applications, the architecture of the data will have a profound influence on the architecture of the software that must process it.

The structure of data has always been an important part of software design. At the program component level, the design of data structures and the associated algorithms required to manipulate them is essential to the creation of high-quality applications. At the application level, the translation of a data model (derived as part of requirements engineering) into a database is pivotal to achieving the business objectives of a system.

Architectural Design Elements

The architectural design for software is the equivalent to the floor plan of a house. The floor plan depicts the overall layout of the rooms; their size, shape, and relationship to one another; and the doors and windows that allow movement into and out of the rooms. The floor plan gives us an overall view of the house. Architectural design elements give us an overall view of the software.

The architectural model [Sha96] is derived from three sources: (1) information about the application domain for the software to be built; (2) specific requirements model elements such as data flow diagrams or analysis classes, their relationships and collaborations for the problem at hand; and (3) the availability of architectural styles and patterns.

The architectural design element is usually depicted as a set of interconnected subsystems, often derived from analysis packages within the requirements model. Each subsystem may have it's own architecture (e.g., a graphical user interface might be structured according to a preexisting architectural style for user interfaces). Techniques for deriving specific elements of the architectural model.

Interface Design Elements

The interface design for software is analogous to a set of detailed drawings (and specifications) for the doors, windows, and external utilities of a house. These drawings depict the size and shape of doors and windows, the manner in which they operate, the way in which utility connections (e.g., water, electrical, gas, telephone) come into the house and are distributed among the rooms depicted in the floor plan.

They tell us where the doorbell is located, whether an intercom is to be used to announce a visitor's presence, and how a security system is to be installed. In essence, the detailed drawings (and specifications) for the doors, windows, and external utilities tell us how things and information flow into and out of the house and within the rooms that are part of the floor plan. The interface design elements for software depict information flows into and out of the system and how it is communicated among the components defined as part of the architecture.

There are three important elements of interface design: (1) the user interface (UI); (2) external interfaces to other systems, devices, networks, or other producers or consumers of information; and (3) internal interfaces between various design components. These interface design elements allow the software to communicate externally and enable internal communication and collaboration among the components that populate the software architecture.

UI design (increasingly called usability design) is a major software engineering action and is considered in detail in Chapter 11. Usability design incorporates aesthetic elements (e.g., layout, color, graphics, interaction mechanisms), ergonomic elements (e.g., information layout and placement, metaphors, UI navigation), and technical elements (e.g., UI patterns, reusable components). In general, the UI is a unique subsystem within the overall application architecture.

The design of external interfaces requires definitive information about the entity to which information is sent or received. In every case, this information should be collected during requirements engineering (Chapter 5) and verified once the interface design commences.8 The design of external interfaces should incorporate error checking and (when necessary) appropriate security features.

The design of internal interfaces is closely aligned with component-level design . Design realizations of analysis classes represent all operations and the messaging schemes required to enable communication and collaboration between operations in various classes. Each message must be designed to accommodate the requisite information transfer and the specific functional requirements of the operation that has been requested. If the classic input-process-output approach to design is chosen, the interface of each software component is designed based on data flow representations and the functionality described in a processing narrative.

In some cases, an interface is modeled in much the same way as a class. In UML, an interface is defined in the following manner [OMG03a]: "An interface is a specifier for the externally-visible [public] operations of a class, component, or other classifier (including subsystems) without specification of internal structure." Stated more simply, an interface is a set of operations that describes some part of the behavior of a class and provides access to these operations.

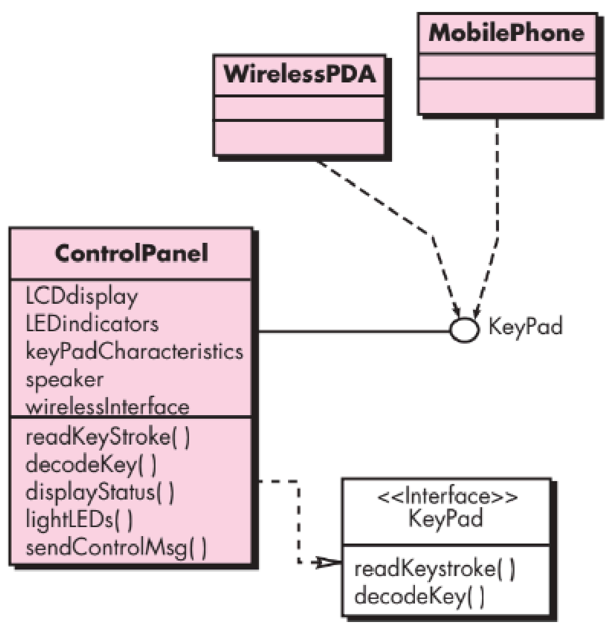
For example, the SafeHome security function makes use of a control panel that allows a homeowner to control certain aspects of the security function. In an advanced version of the system, control panel functions may be implemented via a wireless PDA or mobile phone.

The ControlPanel class (Figure 8.5) provides the behavior associated with a keypad, and therefore, it must implement the operations readKeyStroke () and decodeKey (). If these operations are to be provided to other classes (in this case, WirelessPDA and MobilePhone), it is useful to define an interface as shown in the figure. The interface, named KeyPad, is shown as an <<interface>> stereotype or as a small, labeled circle connected to the class with a line. The interface is defined with no attributes and the set of operations that are necessary to achieve the behavior of a keypad.

The dashed line with an open triangle at its end (Figure 8.5) indicates that the ControlPanel class provides KeyPad operations as part of its behavior. In UML, this is characterized as a realization. That is, part of the behavior of ControlPanel will be implemented by realizing KeyPad operations. These operations will be provided to other classes that access the interface.

FIGURE 8.5

Interface representation for Control-Panel



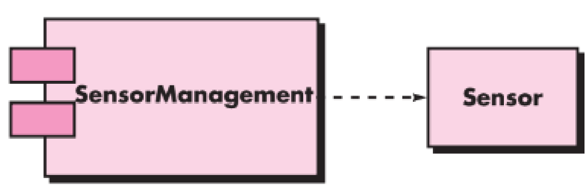
Component-Level Design Elements

The component-level design for software is the equivalent to a set of detailed drawings (and specifications) for each room in a house. These drawings depict wiring and plumbing within each room, the location of electrical receptacles and wall switches, faucets, sinks, showers, tubs, drains, cabinets, and closets. They also describe the flooring to be used, the moldings to be applied, and every other detail associated with a room. The component-level design for software fully describes the internal detail of each software component. To accomplish this, the component-level design defines data structures for all local data objects and algorithmic detail for all processing that occurs within a component and an interface that allows access to all component operations (behaviors).

Within the context of object-oriented software engineering, a component is represented in UML diagrammatic form as shown in Figure 8.6. In this figure, a component named SensorManagement (part of the SafeHome security function) is represented. A dashed arrow connects the component to a class named Sensor that is assigned to it. The SensorManagement component performs all functions associated with SafeHome sensors including monitoring and configuring them.The design details of a component can be modeled at many different levels of abstraction. A UML activity diagram can be used to represent processing logic.

FIGURE 8.6

A UML component diagram



Detailed procedural flow for a component can be represented using either pseudocode or some other diagrammatic form (e.g., flowchart or box diagram). Algorithmic structure follows the rules established for structured programming (i.e., a set of con- strained procedural constructs). Data structures, selected based on the nature of the data objects to be processed, are usually modeled using pseudocode or the programming language to be used for implementation.

Deployment-Level Design Elements

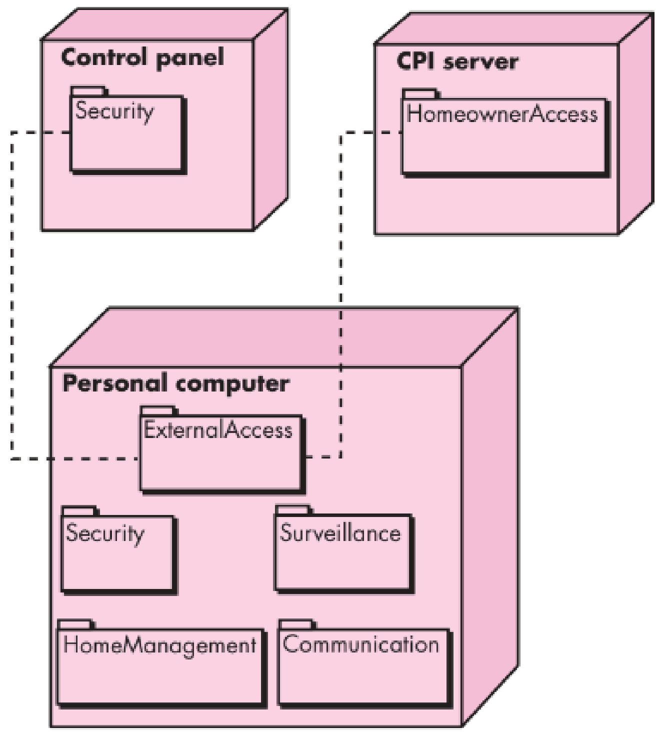
Deployment-level design elements indicate how software functionality and subsystems will be allocated within the physical computing environment that will support the software. For example, the elements of the SafeHome product are configured to operate within three primary computing environments—a home-based PC, the SafeHome control panel, and a server housed at CPI Corp. (providing Internet-based access to the system).

During design, a UML deployment diagram is developed and then refined as shown in Figure 8.7. In the figure, three computing environments are shown (in actuality, there would be more including sensors, cameras, and others). The sub-systems (functionality) housed within each computing element are indicated. For example, the personal computer houses subsystems that implement security, surveillance, home management, and communications features. In addition, an external access subsystem has been designed to manage all attempts to access the SafeHome system from an external source. Each subsystem would be elaborated to indicate the components that it implements.

The diagram shown in Figure 8.7 is in descriptor form. This means that the deployment diagram shows the computing environment but does not explicitly indicate configuration details. For example, the "personal computer" is not further identified.

FIGURE 8.7

A UML deployment diagram



It could be a Mac or a Windows-based PC, a Sun workstation, or a Linux-box. These details are provided when the deployment diagram is revisited in instance form during the latter stages of design or as construction begins. Each instance of the deployment (a specific, named hardware configuration) is identified.