∗ ∗ DRAFT ∗ Please Do Not Copy or Distribute ∗ DRAFT ∗ ∗

Information Engineering: Object-Oriented Design and Analyses*

The Specification Process

Steven A. Demurjian and Heidi J.C. Ellis
Computer Science and Engineering Department
The University of Connecticut
Storrs, Connecticut 06269-3155

August 25, 2012

Contents

Information engineering as defined is comprehensive across the entire design and development process, from the conceptualization of an idea to its final realization and maintenance as a working application. This includes those steps and the necessary individuals for determining the viability and commercialization of a product via techniques such as in-house discussions, market surveys, and cost and feasibility studies. For our purposes, assume that these actions have already occurred and that a 'go-ahead' has been given to develop a product, either for a customer or a perceived target audience. Once this go-ahead has been given, the next, and some argue, most critical aspect of the information engineering process involves what is typically called the *requirements definition* or *specification* of the application.

The specification embodies all of the required features and characteristics of an application. As shown in Figure 3.1, it is a document of many parts and pages. As a document, it must

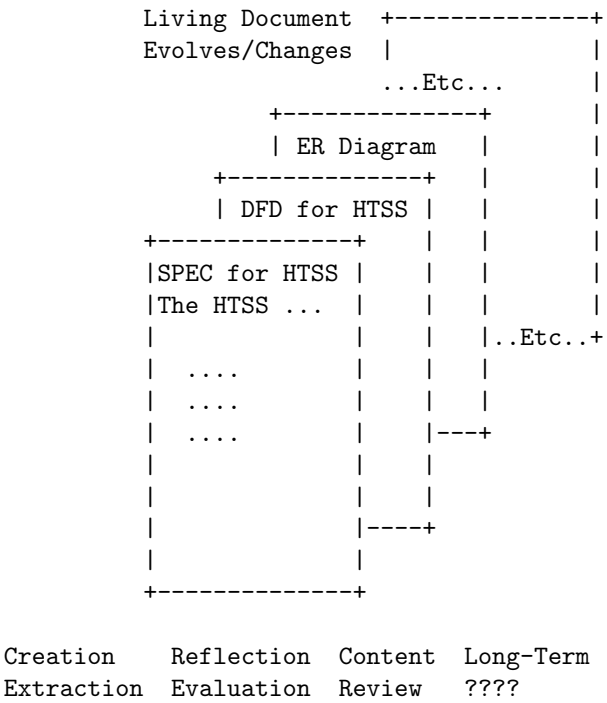


Figure 3.1: What's in a Specification?

be read and understood by a wide variety of individuals (e.g., customers, end-users, designers, engineers, management, etc.), providing breadth and generality on the one hand (for customers, end-users, management, etc.) while not sacrificing technical focus and details on the other hand (for designers, engineers, maintainers, etc.). The specification also serves as a contract between the customer and the company, as well as between the designers and the engineers who are responsible for the development of the application. We can also view the specification as both a *living* and *long-lived* document. The specification is living, since it comes into existence to formalize and delineate an idea and remains in existence throughout the design, development, and maintenance processes. It is long-lived, since these processes can span years (decades?), and changes that occur due to problem corrections or evolutions must always be consistent with the goals of the specification and reflected in its content after the modifications have occurred.

There are many different types and approaches to the content and organization of a specifi-

cation. Ghezzi, et al. [1], distinguishes between formal (verifyable) and informal specifications, and also categorizes specifications as either operational or descriptive in nature. Other techniques such as finite state machines (FSMs), petri-nets (PNs), etc., focus on the functional behavior for an application. Descriptive specifications characterize the features of an application, including approaches such as ER diagrams (the database features), formal logic (constraint features), and algebraic. Our approach to supporting the specification process focuses not only on the available tools (e.g., DFDs, FSMs, ER, etc.), but is geared towards a detailed investigation of the combination of the available tools and techniques into a framework that promotes a more orderly, rigorous, and hence, 'engineering-oriented' development of the many aspects of a specification.

This framework must meet the many features of the specification document as described above. In addition to being both living and long-lived, there are many other features of the specification that must be considered, as reviewed below:

- Content, Context, and Creation: The specification's content (its different sections), context (relationships and interdependencies between sections), and creation (processes used to develop content and context) are critical for developing a well-defined and comprehensive document.

- Extraction and Review: As noted above, the specification needs to provide breadth and generality as well as technical focus and details. For the former, there must be the ability to extract meaningful subsets of the document that can stand alone and are targeted to particular audiences. For the latter, when examined in its entirety, the specification must provide a review of the entire application and/or project.

- Extensibility and Evolution: To support long-lived and living characteristics of the specification, the ability to fix problems and/or increase application functionality are critical.

- Reflection and Evaluation: There is a strong need to have both quantitative and qualitative measures for gauging the effectiveness of the specification process and final document. However, it is unclear the degree to which this feature can be attained.

These features and other characteristics of a specification will be the focus of this chapter. Exercises for this chapter ask you to consider these features in more detail.

A specification must transcend its design or implementation domain to represent the who, what, where, why, when, and how of an application. A good specification will involve and use many of the different operational and descriptive techniques that have been mentioned, with the overall goal to be a complete, robust, and understandable document. A good specification should also be versatile enough to support many different design/implementation approaches (e.g., traditional, object-oriented, functional, etc.).

1 The Individuals Who Use Specifications

The specification contains a characterization of the requirements that must be met in order for the application to be precisely designed and implemented; it continuously changes over time as application requirements are detailed and refined. The specification is utilized as a communication medium and shared focus during its definition process, as a guide and starting point throughout an application's early design stage, as a common context when detailed design and implementation are underway, and as a reference and learning tool for supporting maintenance (both corrective

and evolutionary). Therefore, specifications must be useful to a spectrum of different individuals throughout the entire lifetime of an application.

Consequently, before we undertake the task of detailing the content and components of a specification, we must first investigate, understand, and characterize the many different individuals who require some degree of access to a specification. We have divided our presentation into two broad categories, based on the usage patterns of individuals, i.e., their expertise and their roles within the application development process. We make no claims that our division is "best"; rather, it represents one approach to examining this issue.

1.1 Customers and End-Users

The development of an application can take many forms. Typically, the need for an application is dictated by either a specific customer asking that a particular problem be solved, or a company identifying a target domain that can be served (in a financially advantageous way) by the development of a software system. In either case, there is a perceived *customer*, whether that person is real, or some target group of individuals (e.g., educators, chemical engineers, home-office systems, etc.).

Given this view, customers are the motivating force behind the production of specifications. A customer conceives an idea for a specific system and must convey the characteristics of that idea to an information engineer (IE) who develops a specification using the customer's input and his/her own expertise. Throughout the iterative specification process, the customer uses the specification to understand the intended system's characteristics, merits, and potential shortcomings, and to determine whether the specification meets his/her needs. Since customers may come from non-technical backgrounds (with respect to an understanding of computer science & engineering and not of their own expertise and domain), the understandability of the specifications must be high. In addition, customers may be non-computer experts, so portions of the specifications must be written in prose that is understandable by non-experts.

The *end-users* (targeted group) that will purchase and/or utilize the application can also benefit from the specification. For the naive user, the specification should contain information that can be extracted and reformulated to represent an overview of system functionality and operation. Sophisticated end users can utilize specifications to explore advanced system features. These types of users are interested in exploring the software to its fullest potential, and in some sense, have a similar role to an information engineer that designs and/or develops the application. In both situations, to fully exploit some of the more subtle aspects of the application, users may access specifications (or, at the very least, a document derived from it) to identify and clarify the detailed abilities of the system.

1.2 Software Engineers

Our motivation and focus throughout the book is the engineering of information. Thus, rather than refer to individuals that specify, design, and develop applications as software engineers, we will employ the more powerful term *sofware engineers (SWEs)*. SWEs transcend the capabilities of software engineers, since their responsibilities are not just limited to developing software. They must also interact with customers and end users, specify and design systems, and maintain systems after development (which also requires interaction with dissatisfied or frustrated end-users). SWEs *engineer information*, since they must collect, decipher, interpret, absorb, and disseminate information as part of their roles. The technical expertise and responsibilities of SWEs are diverse and broad. To meet these varied needs we define the following types of SWEs:

Software Specification Engineer (SSE): SSEs are responsible for the creation of specifications. They take customer ideas as input and construct a specification that meets the customer's requirements. ISEs must also target the specification to the many different audiences (e.g., customer, IDEs, IDevEs, etc.) that will need access to it over its lifetime. ISEs utilize previously constructed specifications to aid in the construction of current specifications as well as to evaluate the completeness and correctness of specifications under construction.

Software Design Engineer (SDE): SDEs provide the bridge between specifications and implementation by producing a detailed system design. IDEs rely heavily on specifications when developing the application. Specifications allow IDEs to explore design issues, evaluate design alternatives, and determine the correctness of the application's design at different stages in the design and implementation process. Specifications are consulted closely during each step of application development to ensure that the requirements are met.

Software Development Engineer (SDevE): SDevEs convert the detailed design produced by the SDEs into the application by implementing the design for some target hardware/software environment. SDevEs consult specifications and designs to ensure that the specifications are met by each portion of the application as it is developed. SDevEs also evaluate the completed application to determine that all requirements are met.

Software Management Engineer (SManE): IManEs use specifications to aid in the guidance and management of an application's development. Specifications provide information regarding the amount and kind of resources (e.g., tools, PLs, hardware, etc.) required to design, develop, test, and maintain the application. This information is utilized by managers to set up schedules, plan release dates, predict required man-hours, and estimate costs. The separation of concerns within the specification allows work to be more easily partitioned, allocated, and scheduled among a team of engineers.

Software Maintenance Engineer (SME): Specifications are used by SMEs to ensure that both bug fixes and evolutionary changes made to the application meet the specification requirements. In turn, error corrections and adaptive changes made to an application require that the specifications and designs be updated to reflect such modification, thereby insuring information consistency.

Other Information Engineers: Specifications are also used as an example for other information engineers at a company. Newly hired information engineers use the specifications to aid in the understanding of how specifications are created. Specifications also help these engineers become familiar with a company's current products. Experienced information engineers reuse or customize specifications for other, similar applications. Specifications are evaluated by experienced engineers to determine how the specifications facilitated the system development process.

We do not intend this to be an exhaustive list of all possible SWEs. Rather, it is a starting point for understanding the role and responsibility of individuals as related to the specification process.

2 The Organization and Content of Specifications

In general, the specification (or portions of it) must be general enough to support the principle of *anticipation of change*. That is, as a living document that exists over months and years, the specification is always changing. On the other hand, the specification must be comprehensive and detailed enough to allow the correctness, performance requirements, portability, interoperability, and reusability of the application to be easily characterized and verified. The 'completeness' of a specification is also important, but, an open-ended design precludes its attainment. However, we must still strive for a specification that is as 'complete' as possible, while acknowledging that this is impossible to evaluate and validate with any degree of certainty. The use of more formal specification languages ensures rigor and formality in the specification, increasing the maintainability and productivity of the specification. But, such formal specification languages are inadequate when large-scale and complex applications must be developed.

The format of the specification described below is intended to meet the needs of the variety of individuals who use the specification, as we discussed in the previous section. It is detailed and specific enough to be used by information engineers for application design, implementation, and maintenance, while also providing sufficient general information to allow non-computer experts (such as customers, end-users, etc.), to gain an understanding of the application and its intended operation. The principle of *separation of concerns* (i.e., clearly identify the facets of any problem to be solved) is used to organize the specification into sections where each section addresses a different aspect of the application to be designed. Some sections are closely related to other sections, and therefore either contain overlapping information and responsibility, or they must be designed/developed in parallel due to their interdependencies.

Our presentation on the content and organization of the specification is broken down into two parts. First, the requirements of the application with respect to its intended functionalities, characteristics, and domain, are described. Then, the issues that relate to the actual development of the application (the engineering of the application), the environment in which the application is created, and methodologies/approaches to be used during development, etc., are discussed.

2.1 Application Issues and Requirements

In our efforts to date, we have identified a specification document that contains a total of eight sections to describe the scope and breadth of an application's requirements. The titles of these sections are:

1. Introduction
2. Glossary
3. Operating Environment
4. Interfaces
5. Interfaces
6. Information
7. Performance
8. Security

To explore each of the sections in more detail, we will employ a uniform and consistent presentation. Specifically, for each section, we will indicate its general definition and purpose, provide a list of questions that can be used to identify and characterize the application, followed by a further discussion and explanation, and completed with an example from the High-Tech Supermarket System (HTSS). Also note that, while we have ordered these sections as presented, our organization has been influenced by our desire to have the structure of the final document occur in a specific way. We acknowledge upfront that constructing the specification is not a sequential process, it is iterative and cyclical, occurs differently for different application domains, and the final document itself might have a better presentation order as suited to particular needs.

2.2 The High-Tech Supermarket Example

The High-Tech Supermarket System, HTSS, is a modern supermarket that uses the newest and most up-to-date computing technology to support inventory control and to assist customers in their shopping experiences. The purpose of HTSS is to utilize computing technology in a positive way to enhance and facilitate the shopping experience for customers. The chain wants to integrate inventory control with:

1. the cashier's functions for checking out customers to automatically update inventory when an item is sold,

2. a user-friendly grocery item locator that indicates textually and graphically where items are in the store and if the item is out of stock, and

3. a fast-track deli-orderer (deli orders are entered electronically), with the shoppers allowed to pick up the order weighed and packaged without waiting.

The inventory control aspect of the proposed system would maintain all inventory for the store and alert the appropriate store personnel whenever the amount of an item drops to its reorder limit. In addition to the aforementioned functional characteristics, the system should also have extensive query capabilities that allow store personnel to investigate the status of the inventory and sales for the store over varying time periods and other restrictions. Finally, note that HTSS and its functional components are based on an actual store, Super Stop & Shop, that opened in Manchester, Connecticut, in the spring of 1993. Thus, the concepts that are presented have their basis in an actual 'real-world' application.

To support the functional and operational requirements of HTSS, from an end-user perspective, there must be a set of user-system interfaces. Possible interfaces include:

- Cash Register/UPC Scanner: Used to process an order, which includes: recording individual items, totaling them, deducting coupons, and taking payment. As each item is scanned, it must be deducted from the inventory, so that values are always consistent and up-to-date.

- Displays for Inventory Querying: To access and manage the inventory, a separate display is needed. Through this display, orders and updates can be made. Only authorized individuals will be allowed to enter new orders or update the inventory when a shipment arrives.

- Shopper Interface for Locator: Used by customers to locate where (aisle, shelf) a particular item is displayed in a store.

- Shopper Interface for Orderer: Through this interface, customers can place orders for the deli (e.g., meats, cheeses, salads, etc.). These orders are then filled and the customer picks up the order at some later time.

- Deli Interface for Orderer: This interface is needed by store employees that work in the Deli department to scan and fill customer orders.

We make no claims that these are the only user interfaces; rather they are the ones which are discussed in examples throughout this and later chapters. We have chosen this set based on both their differences (they all have unique requirements for their operation) and similarities (they all share common requirements regarding response-time, throughput, and user-friendliness). Response-time and throughput are important for the first two interfaces, since there is likely to be

multiple cash registers that must work in parallel with many inventory displays. User-friendliness is also important, for new employees using cash registers, and especially for customers using the different shopper interfaces.

2.2.1 Introduction

The *introduction* provides an overview/problem statement of the application's domain and purpose. It is intended to provide the non-expert reader with an understanding of the general purpose, scope, and utility of the application. The questions that the reader wants answered at this point is *What does the application do?* and, for customers that may be interested in purchasing a completed system, *Will it fit into my business and run on my hardware?*. In particular, the introduction should contain answers to the following questions:

1. What is the domain for the application?
2. What is the profile of intended users/companies for the application?
3. What is the main purpose of the application?
4. What is the scope/range of the application?
5. What are the application's major functionalities?
6. What is the target execution environment (e.g., embedded, PC, custom-hardware, etc.)?
7. What are the critical user/system interactions?
8. What is the general form of user interaction with the application (i.e., graphic, textual, etc.)?
9. Are there any unique hardware/software requirements?
10. What are the application's supported interfaces to other software packages?

The introduction should contain a description of the domain to which the application applies and a clear identification of the objectives of the application. The introduction should also include a brief discussion of the intended users of the application and the environment (stand-alone, networked, PC, etc.) in which the application is to be used. The introduction should be easily understandable by the general public, a prose description that can be utilized by both system designers and non-experts.

A skeleton of the introduction for the HTSS application, as shown again for completeness below:

> **Introduction**
>
> The High-Tech Supermarket System, HTSS, is a modern supermarket that uses the newest and most up-to-date computing technology to support inventory control and to assist customers in their shopping experiences. The purpose of HTSS is to utilize computing technology in a positive way to enhance and facilitate the drudgery of shopping for customers. The chain wants to integrate inventory control with:
>
> 1. the cashier's functions for checking out customers to automatically update inventory when an item is sold,

2. a user-friendly grocery item locator that indicates textually and graphically where items are in the store and if the item is out of stock, and
3. a fast-track deli-orderer (deli orders are entered electronically), with the shoppers allowed to pick up the order weighed and packaged without waiting.

The inventory control aspect of the proposed system would maintain all inventory for the store and alert the appropriate store personnel whenever the amount of an item drops to its reorder limit. In addition to the aforementioned functional characteristics, the system should also have extensive query capabilities that allow store personnel to investigate the status of the inventory and sales for the store over varying time periods and other restrictions

To support the functional and operational requirements of HTSS, from an end-user perspective, there must be a set of user-system interfaces. Possible interfaces include:

- Cash Register/UPC Scanner: Used to process an order, which includes: recording individual items, totaling them, deducting coupons, and taking payment. As each item is scanned, it must be deducted from the inventory, so that values are always consistent and up-to-date.
- Displays for Inventory Querying: To access and manage the inventory, a separate display is needed. Through this display, orders and updates can be made. Only authorized individuals will be allowed to enter new orders or update the inventory when a shipment arrives.
- Shopper Interface for Locator: Used by customers to locate where (aisle, shelf) a particular item is displayed in a store.
- Shopper Interface for Orderer: Through this interface, customers can place orders for the deli (e.g., meats, cheeses, salads, etc.). These orders are then filled and the customer picks up the order at some later time.
- Deli Interface for Orderer: This interface is needed by store employees that work in the Deli to scan and fill customer orders.

Some further requirements are related to response-time/throughput and user-friendliness. Response-time and throughput are important for the first two interfaces, since there is likely to be multiple cash registers that must work in parallel with many inventory displays. User-friendliness is also important, for new employees using cash registers, and especially for customers using the different shopper interfaces. To support all of the information requirements of HTSS, a number of interacting and interdependent databases will also be required.

While this introduction may not satisfy every question given above, it would serve as an excellent starting point for a more detailed and expansive introduction for this application.

2.2.2 Glossary

The relevant and important terms used throughout the other sections in the specification are explained in the *glossary*. This section is intended to reduce potential confusion, and to increase the correctness, robustness, and verifiability of the specification by clearly defining all critical terms used in the specification. Some terms may have more than one definition depending on the reading audience. The following questions should be resolved by the Glossary section:

1. What are the relevant terms used throughout the specifications?
2. What are the special terms that must be defined and understood to: read, comprehend, characterize, and describe the application?
3. What terms could be interpreted ambiguously and what do they mean in the context of the specification?
4. For each "term", what is its scope of importance with respect to the application, its functionality, and its operation?
5. Which terms, if any, apply to a particular portion of the application?
6. Which terms are required by each of the different types of engineers?

The last question pertains to the construction of the specification, rather than the specification itself. Note that the definitions of terms give the reader some idea of the purpose and operation of the application. That is, the information provided by the glossary goes beyond the simple meaning of the terms. Definitions are prose and may include descriptions of information required, its usage, and/or the operation of some portion of the application as needed to fully define the term.

In the HTSS application, there are a number of obvious terms that would comprise the initial version of the glossary.

Glossary

Item: Refers to a product that is sold at a supermarket, e.g., canned goods, cereals, meat and deli items, produce, etc.

UPC: The Universal Product Code for all items in a supermarket.

ICDB: The Inventory Control Database, functions as a central repository that tracks the features and characteristics of all items on shelves and in the stockroom.

I-Controller: An I(nventory)-Controller is the individual at the supermarket who is in charge of maintaining the inventory by reordering stock for Items that fall below a reorder limit.

Courtesy Card: A Check Cashing Card that has been assigned to a shopper. This indicates that the credit has been established for a shopper and his/her checking account.

PIN: A Personal Identification Number used by a shopper when paying for groceries with a Debit (Bank) Card.

Notice that some of the defined terms refer to individuals and end-users. It is important to note, that as the specification evolves, and other sections are defined and revised, the glossary must be constantly updated to reflect the changes.

2.2.3 Operating Environment

This section details the *operating environment* in which the application is to be run. The following list of questions highlights the issues that should be resolved in developing this section of the specification:

1. What will be the expected hardware, software, and operating environment of the application?

2. What, if any, are the off-the-shelf software systems/tools that are needed for the application?

3. What is the physical environment (e.g., location, temperature, etc.) that the application will exist in?

4. If the environment is hostile, what additional safeguards must be considered?

5. What types of users will utilize the application (e.g., naive, expert, etc.)?

6. What is the expected usage pattern for the application? or, in other words, What are the high/low times and number of users?

A prose description elaborates on the intended hardware, its configuration, and support software. Included in this section is a description of any networks the system will be expected to operate on including number of nodes and locations as well as any other requirements of the run-time environment.

There are many examples of systems that require specialized hardware, displays, and/or software. For example, related to question 3 above, many service stations allow customers to pay for gasoline at the pump by using a credit or debit card. Since pumps operate in an open environment with at best a minimal roof or covering, the physical environment must be identified and considered as part of the specification (and eventual design and development). ATM machines tend to avoid this problem, since they often operate inside an entry way of a bank. However, ATMs are in a hostile environment (re. question 4) since they are prone to vandalism and robbery.

In our HTSS, there are many specialized considerations that must be identified during the specification process. Some of the more obvious ones are elaborated below:

> **Operating Environment**
>
> The operating environment for the HTSS application will involve a collection of interconnected terminals. There are five different kinds of terminals: cashier (to total orders), deli-orderer (for shopper), deli-display (for deli workers to process orders), item-locator (for shopper), and inventory control. The two deli displays and the locator must have sealed data entry keys to insure that spills, dirt, etc., will not inhibit operation. The cashier display needs two interfaces to track the items and their costs for both the cashier and shopper. Cashiers and inventory workers will be trained to use their displays, and the input will use either keyboard or mouse technology. To allow deli workers to keep their hands free, a voice controlled interface must be developed to allow orders to be processed. Finally, since shoppers are "naive" users, the interface for the deli-orderer and item-locator must be geared towards their skill levels. The expected throughput for each cashier at peak levels is 10 customers per hour with a maximum of 20 cashiers. There are two deli-orderers and five item-locators that must ...etc...

This example illustrates that a great deal of interaction is required with the customer to understand and describe the operating environment and its constraints. Note that we have just begun to scratch the surface regarding performance issues for the application. In this particular case, we are also interested in the average number of items per order, since this has a direct impact on database accesses, which in turn affects overall throughput and system performance.

2.2.4 Interfaces

A thorough description of all *interfaces* to the application, including graphical, user, inter-system, and database interfaces, are detailed in this section of the specification. The intent is to provide an understanding of the application's usage and to allow the reader to gauge, evaluate, and realize the interoperability and portability of the system. Relevant questions that should be considered in this section include:

1. What are the major user and/or system interfaces (including those for the database)?
2. What is the goal and purpose of each interface?
3. What are the major functions and responsibilities of each interface?
4. What is the nature of each user interface (e.g., graphic, text, etc.)?
5. Who uses (with respect to expertise) each interface?
6. What information units must be provided to each interface?
7. What information units are received from each interface?
8. How does each interface manipulate database information?
9. What are the dependencies/interactions among interfaces (with respect to currency of displays, information consistency of updates, etc.)?
10. What are the restrictions on each interface (i.e., are there conditions under which the interface will not operate)?
11. What are the information application requirements for each interface? (e.g., hardware interface may need a baud rate, parity, etc.)

This section of the specification contains a discussion of every interface (i.e., user, software, hardware, and database) of the application, including its expected behavior and any requirements/restrictions. The interaction between the interfaces and the database is also detailed in this section. For each interface, a description of which portions of the database are accessed, displayed, and/or modified (i.e., insert, delete, or update) is included. The form of the descriptions may be either natural language or, in the case of graphic interfaces, a graphic diagramming of the appearance of the desired interface. High-level ER diagrams may be included to describe information that is required to be stored and retrieved through a database interface. These descriptions are intended to provide the reader with an understanding of the behavior of each interface without detailing how the behavior occurs. The descriptions are broad in nature to support the principles of generality and portability.

Continuing with the HTSS example, there are a number of obvious interfaces to the system, related to the required graphical displays and data entry for a cash register, deli-orderer, item-locator, deli-display, and inventory control. Note that these displays are directly related to the operating environment components discussed in the previous section. While these displays correspond to the major user/system interfaces per question 1, and their goals, functions, users, and makeup (questions 2, 3, 4, and 5, respectively), can be easily detailed, a major consideration that must also be addressed is information consistency. Recall from Section 1.4 of Chapter 1 that information consistency has four characteristics, as related to the usage, persistence, integrity

and security, and validity or relevance of information. These characteristics are directly related to questions 6, 7, and 8. To handle these issues, interactions with the database must be detailed, as discussed below:

Interfaces

Collectively, the interfaces for the cash register, scanner, deli-orderer, item-locator, deli-display, and inventory control, as described in the introduction, require the following database interfaces:

ICDB: A database of all items is maintained, with each interface requiring a different level of access. All of the user/system interfaces require read access to accomplish their respective tasks, e.g., the scanner must verify the UPC code of an item with its database entry. In addition, the inventory control interface will need to modify the database when items are reordered, while the cash register interface will issue a command to decrement stock on the shelves when an item is sold. This decrement operation is important, since it impacts on consistency and ensures that the item-locator has up-to-date information.

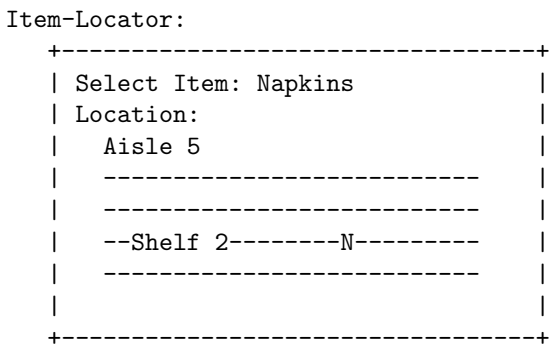
Order DB: A database is required to track individual customer orders. The interface to this database is required by the scanner and cash register interfaces in ringing up and totaling an order. Write access to create an entry in the Order DB is necessary.

Deli-Order DB: A short-term persistent database is required to hold orders entered via the deli-orderer for processing by the deli-display. The interface to this database requires both read and write access by the two aforementioned user-system interfaces.

Note that, ER diagrams similar to the one given in Figure 2.3 of Chapter 2, are appropriate for characterizing the basic database structure of the information units and their interfaces to the other non-database displays. Once the individual functions of each interface have been described, their interactions and interdependencies must be considered as noted by question 9. In this example, clearly the ICDB and Order DB must be consistent, in the sense that the orders for a given time period on all items must coincide with a snapshot of the inventory on each side of the time period. The remaining two questions deal with operating environment issues, again indicating the overlaps within the specification.

In defining interfaces for an application, like HTSS, it may also be feasible during subsequent iterations of the specification to provide mock-ups of the different user interfaces. For example, in HTSS, likely user interfaces include:

```
Deli-Orderer:
   +----------------------------------+
   |Select Item: Ham    Quantity: 1lb |
   |Select Item: Swiss  Quantity: 2lbs|
   |     ...                          |
   |End Session (Y/N): Y              |
   |Order Id: 1234                    |
   |                                  |
   |  *** Pick up in 20 minutes ***   |
   |                                  |
   +----------------------------------+
```



The degree of complexity for mocking up user interfaces is limitless; drawing tools available on PCs and workstations, as well as user interface toolkits to combine primitives into a working prototype, are all available to provide the customer with the intended look-and-feel of the system. This allows alternatives to be considered and substantial changes to be made without a significant impact (as is often the case when this occurs post-design or post-implementation).

2.2.5 Information

The *information* required by the application is clearly identified and described in this section. Major units of information are identified and each unit of information is determined to be either persistent (i.e., stored in the database) or temporary (i.e., not stored in the database). The following questions should be resolved in this section:

1. What are the major units of information used by the application? By each interface? By each "state" of application operation?
2. For each unit of information, elaborate on:
   - its purpose in the application
   - its lifetime (persistent or not)
   - its usage or intended usage
   - its availability to various system components and interfaces
   - its accessibility by different end-users
   - its general semantic structure
   - its interactions/interdependencies with other units of information
   - its independence with respect to other units of information
   - its structural/behavioral relationship with other units of information
   - its storage/access in the database

Clearly, the details presented in the Interfaces section has begun to address the many issues related to the information, its usage, and its consistency. Specifically, for question 1, we have already detailed a number of the requisite information units by naming three databases (i.e., ICDB, Order DB, and Deli-Order DB). This section also addressed many of the issues from question 2. However, the difference between the Interface section (and its consideration of information) and this section is one of granularity. Here, our focus is on detailing the gross/high-level structure of each database, rather than naming particular databases. To meet this goal, an ER diagram

similar to Figure 2.3 for HTSS in Chapter 2 is apropos. Recall that in that diagram, our focus was on the major units (i.e., the entities) of information, their composition (i.e., their attributes), and their interactions (i.e., the relationships including inheritance) with other units. This definitely fits the requirements of the second question.

If we examine the requirements of this section of the specification more closely, it should be apparent that the major goal is for the ISE to identify the general purpose and type of each information unit using a natural language, ADT, or ER-like approach. Another important facet to consider in this process has been described in [1], where information units are classified as either an information source, information producer, or information maintainer. This classification is important, since it indicates how the different information units are utilized within the application. Given a high-level conceptualization of the major information units, separate subsections for each information unit should be provided and contain: a description of the type of the unit, an explanation of its intended purpose, and an indication of which portion(s) of the application (and in conjunction with which other major information units) that the unit is used. ISEs and IDEs may add further detail during the design process by creating pertinent DFD diagrams that demonstrate the intended use of the unit, which may overlap with those found in the Operation section, and ER diagrams to depict the internal structure and inter-relationships to other units, which may overlap or refine ER diagrams found in the Interface section. In addition, for each unit of permanent information identified, an explanation of the expected access usage and any other database system considerations (e.g., clustering, concurrency, recovery, etc.) are included.

The purpose of the information contained in each subsection is two-fold: to provide the system designer with explicit knowledge about the information requirements of the application; and to supply sufficient information to aid in the construction of a *profile* associated with each major information unit in the application. Profiles contain information about the name, character, and purpose of each unit of information. These profiles are used during design to facilitate different levels of analyses over the entire application. The profile concept is very critical and is used heavily in later chapters of the book.

2.2.6 Performance

This section of the specification addresses the *performance* requirements of the application, with the intent to clearly define the performance parameters under which the application must perform. Overall system performance has already been addressed in part by the Operating Environment section. However, remember that this focus was from the customer/application perspective, i.e., orders per cashier, maximum number of cashiers active, etc. We must now bridge the gap from expected performance of the application in its final environment by considering the detailed performance of its overall structure and components, so that more general system performance levels can be met. The following questions can assist in this process:

1. What, if any, are the performance or response time requirements for the overall application?
2. What type of users will access the application?
3. What are the maximum (minimum) number of users?
4. What is the expected usage pattern?
5. What is the tolerance for error (video game vs. flight controller)?
6. What is the expected workload pattern for networked application?

7. What resources and which of these resources are "critical"?

A general prose discussion of performance requirements details the usage for the application including the expected maximum and minimum number of users, average usage patterns, and tolerance for errors. The tolerance for errors is an interesting aspect of this portion of the overall performance, since it deals with fault tolerance and relevance issues on information. For example, in our HTSS, the impact of inconsistency when a shopper uses the item-locator, and upon reaching the shelf finds that the item is out of stock, is not as critical as in some other applications. The shopper may become annoyed, but as long as this inconsistency is infrequent (e.g., once an hour?), it is unlikely to be a major issue. In fact, until the problem becomes pronounced enough that many shoppers begin to bother the store manager, or in fact leave to go to another store, then such a scenario is really a non-problem. However, if the application is an embedded program that controls targeting of missiles on a jet fighter, or a life-support monitor in a hospital emergency room, such a consistency lapse is not acceptable! Clearly, such important operating constraints must be mentioned as part of the introduction, operating environment, and system operation.

More specific performance requirements for the application are addressed in three parts, as related to the interfaces, system structure, or databases.

Interfaces: The following questions pertaining to the performance of the application's interfaces should be resolved in this section:

1. What is the expected response time for each interface?
2. What is the expected refresh time from each user interface?
3. Under what conditions may the interface fail?

This section contains prose descriptions of the performance characteristics (e.g., expected response times, refresh times for all user interfaces). Mathematical formulas may also be included.

In our continuing example, the performance of interfaces for the scanner, cash register, deli-orderer, and item-locator are all critical. In the case of the scanner, the time between the recognition of the UPC code, the checking of the code against the ICDB, and the generation of an entry on the customer's receipt must all occur in what seems to be an instant. Otherwise, both shoppers and cashiers will become quickly discouraged if the check out process is slow. Both the deli-orderer and item-locator must also have reasonable response times, since a computer interface that takes too long for the customer will not be utilized extensively. In developing this portion of the specification, care should be given to carefully enumerate all of the detailed interface performance requirements.

System Operation: The timing requirements of the internal (non-interface) operation of the application are detailed in this subsection. Pertinent questions include:

1. What portions of the application are critical for producing the required response times?
2. What portions of the application do not affect response time?
3. What communication is required between components and what is its affect on performance?

Included in this section is a discussion of any limitations on the operation times and size for any critical portions of the application. These limitations are likely impacted by the

performance requirements of both the interfaces and databases. In the case of the interfaces, for HTSS, the scanner must be able to scan and send information to the cash register. All aspects of the application involved in this process must be carefully developed to consider this important and time-critical action. Since a database access (or two) is likely required, this must also be considered as part of the overall performance with respect to system operation.

Databases: Requirements for the database such as the size, response-time, and throughput are described in this section. Pertinent questions include:

1. What is the expected size of the database?
2. What is the expected response time/throughput for the database? (Minimum and Maximum ranges)
3. What are the access patterns and response-time needs of the database?
4. How is information consistency maintained as related to propagated, persistent, and/or temporary information?

One of the issues that must be addressed as part of the performance of the database is its distribution. For example, if all scanners and cash-registers must consult a single, shared, centralized repository of data (the ICDB and Order DB), there is a potential for a serious system bottleneck. Even though both of these interfaces use the ICDB in a read-only fashion, this is not likely to be a major problem, since price updates (either wholesale or retail cost) and UPC changes happen either infrequently or in a controlled fashion (e.g., Saturday at midnight for Sunday AM sales). However, the Order DB must be both read and written by both interfaces. Thus, the issue to consider related to performance is whether this database can be distributed to reside locally at the register (while the order is totaled) and then migrate to permanent storage after payment has been received. Since distribution is an issue, the parallel and concurrent engineering features of the application are dependent and inter-related to this portion of the specification.

The above discussions can take many different forms and may include mathematical functions to describe performance behavior. General control-flow graphs (CFGs), queueing models, and simulations may be constructed for each of the above sections by the ISE, while the IDE may create more detailed CFGs to demonstrate the expected performance behavior of the application.

2.2.7 Security

This section details the *security* requirements for the application. A security plan for the application is required to ensure that unauthorized personnel do not obtain access to restricted information, and that authorized personnel are not prohibited from accessing required information. Collectively, positive (prohibited) access can be used to indicate the sharing (independence) that must be supported. When developing this final part of the specification, a general explanation of the intent of the security approach must be initially included to clarify the motivation behind the approach. Once this has occurred, all of the potential users of the application are organized into types of users (e.g., store managers, deli orderers, cashiers, etc.) according to their responsibilities within the application. We call each of the types of users a *user role*. The following questions should be resolved in the Security section:

1. What is the general intent (overview) of the security plan?

2. What types of users (user roles) will access the application?
3. What are the responsibilities for each user role?
4. What information units must each user role access?
5. What information units must each user role be restricted from accessing?
6. Which user roles are restricted from using which interfaces?
7. For each information unit, which user roles are allowed read and/or write access?

After a general discussion of the security requirements has been developed, the remainder of this portion of the specification focuses on the relevant user roles. A subsection for each user role contains a prose description of that user role, a description of the responsibilities that correspond to the user role, and an explanation of the security requirement associated with the role. The security requirement description is an explanation of the access that the user role has to the information units identified in the Information section. In addition, this section contains a discussion of the use of the database from a security perspective. The discussion includes an explanation of which portions of the database can be accessed, displayed, or modified by a particular user role and an explanation of what information must be prohibited from use.

To illustrate the user-role concept, consider once again the HTSS application. The following excerpt indicates an appropriate first attempt at defining the security for the application:

> **Application Security**
>
> In HTSS, the main security policy is to encapsulate the majority of functions that write to a database into the design of the different interfaces and system components. While the scanner and cash-register must write the Order DB, this write action should not be the result of an explicit command entered by the cashier. Rather, it should be a responsibility of the software that implements the two interfaces. The item-locator must be limited to read access of the ICDB, with the deli-orderer and deli-display allowed to modify the database. In the case of the deli-orderer, the tool controls the creation of database entries that represent customer orders. For the deli-display, the deli worker must physically issue a command that deletes the order from Deli-Order DB once it has been filled. Finally, the inventory controller is given the most access, since I-Controllers must have the ability to create, modify, and delete Items from the ICDB.
>
> **User Roles**
>
> **Cashier:** Responsible for ringing orders using the scanner (to ring items according to their UPCs) and the cash register (for totaling the order and collecting payment).
>
> **Shopper or Customer:** Buys groceries from the store and uses both the deli-orderer and item-locator interfaces.
>
> **Deli-Display:** Contains read and write access to the Deli-Order DB for processing orders from shoppers and removing orders after they have been filled.
>
> **Store Manager:** Has access to all aspects of the application and its interfaces, and can function as a super-user to override any restrictions.
>
> **Order-Item:** The steps and actions taken by a manager, an I-Controller, or automatically by the system, as related to ordering out-of-stock items.

Receive-Item: The steps and actions that are required whenever a delivery occurs, e.g., check the stock, add to the inventory, stock on shelves (if necessary), etc.

Notice that user roles are not restricted to individuals. The different interfaces that are critical for the application also have roles. This is due to the fact that when each of these interfaces have been successfully implemented, we must insure that their resulting functionalities do not conflict with their intended responsibilities as related to their user roles. Note also that user roles such as Order-Item and Receive-Item are detailed and specific activities that could fall under the responsibilities of other user roles like Store Manager or I-Controller. So the user role is a powerful concept that can control access to information for maintaining consistency and for guiding application development.

2.3 Application Development Requirements and Issues

This section contains a description of the environment in which the application is to be developed. This issues has been heavily addressed by other authors in traditional software engineering textbooks, so we provide only a brief discussion herein. However, we do note that other sections of the specification (i.e., Introduction, Operating Environment, System Operation, etc.) will all impact on the decisions made for the implementation environment. Relevant questions include:

1. Who are the design/development team members and what are their responsibilities?
2. What language(s) (programming, database, other) will be used to implement the application?
3. What software/hardware tools will be used to implement the application?
4. What is the hardware on which the application will be developed?
5. What is the expected release date?
6. What is the schedule for application development?
7. What software libraries (must/should) be used? (commercial, in-house, etc.)
8. Will there be any code reuse? What is it?
9. What company policies must be followed during development?
10. What general methodologies will be followed during development?
11. What management approach will be used to guide development?
12. How is the work structured/divided?
13. What sort of testing/verification techniques will be used?

A prose description elaborates on the hardware and software systems that will be used to construct the system, the schedule by which the application must be completed by, any company policies or preferred methodologies that direct the system development processes, and any languages/tools that are to be employed in the creation of the application. For the HTSS, the available hardware technology for the scanner, cash register, item-locator, deli-orderer, and database management will all impact on the choices made in this section.

3 The Health-Care Application

This section contains a partial specification of the Heath-Care Application (HCA) for use in examples throughout the remainder of the book. This section, and the one that follows related to CAD, also serves as a guide for SWEs who are interested in exploring and developing specifications for their own applications. HCA was motivated and developed on the basis of a call by former Health and Human Services Secretary Louis Sullivan in 1992 for a national database for patient records and claim form processing. This is also related to the current debate regarding a national health care plan for the USA. For HCA, some individuals indicate that the level of complexity of such a system has not be seen in previous applications. The potential involved entities: hospitals, insurance companies, MD offices, government agencies, patients, universities and research institutes, medical supply and pharmaceutical companies, local drugstores and pharmacies, etc.; are numerous, and are all driven by unique and at times, conflicting needs and requirements. The information for such an application is staggering, both in volume (10s of thousands of gigabytes or more?) and in complexity (textual, record-based, x-rays, images, etc.). Even if we could support the information needs, there are many ethical questions regarding protection and accessibility of patient information, that were discussed in Chapter 1. In addition, the technical issues related to standards for patient data and claim forms, and the integration and interoperability of existing heterogeneous hardware, software, and communications systems are monumental problems with no obvious solutions. The specification presented herein is based on our earlier efforts [2, 3], and was influenced by examples in [4].

Introduction for HCA

A system for supporting all aspects of health care delivery is required as one of the critical steps towards a national health care policy. The goals of such a system are:

1. Integration of all delivery units of HCA: e.g., hospitals, private laboratories, physician offices, insurance companies, patient data, etc.

2. Information consistency (usage, persistence, relevance, and integrity/security) is foremost in HCA.

3. Integration provides a seamless environment that standardizes and automates claims processing and billing.

4. Longitudinal data on different aspects of health care delivery can be used to evaluate performance and predict trends, e.g., a patient's records can be analyzed to predict the most likely future problems and employ preventive medicine to anticipate their occurrence and reduce their impact.

5. Summary data on different illnesses, available across the country, can be utilized as the basis of treatment and research studies. For example, the Orphan Drug Act provides monetary incentive for pharmaceutical companies to develop drugs and treatment plans for illnesses that affect only a small subset of the population. Despite this Act, there are still difficulties in identifying patients who have a specific illness. A comprehensive repository, such as the one needed for HCA, could serve such a purpose.

One of the most important aspects of HCA involves the potential of misuse of information, i.e., confidentiality is critical! For example, a patient's history might be used to deny insurance or

to disclose an illness that could impact significantly on his/her lifestyle. Other issues related to HCA are:

- Wide variety of users both across delivery and within a subset, e.g., hospitals have nurses, MDs, technicians, administrators, housekeeping, etc., who all require some degree of access to HCA.
- Interoperability is critical due to myriad of existing hardware/software, e.g., hospitals have separate systems for laboratory analyses/results, scheduling, patient classification, budgeting, etc.

This initial characterization of HCA serves as only the basis for detailing and elaborating on the overall system requirements and capabilities.

Operating Environment for HCA

The operating environment for HCA will need to tackle many issues related to interoperability and usage, including:

- Within each specific delivery unit (e.g., hospital, MD office, etc.), all of their current systems must be integrated with the common system that supports HCA. For example, in a hospital, the system that processes certain laboratory tests contains typical computing hardware and software, and also includes specialized hardware for storing and analyzing fluids to determine test results. These results would need to be integrated with a local hospital patient care database to allow the transmission of time-critical laboratory tests.
- Within each specific delivery unit, varied and specialized interfaces are required. For example, in a hospital, relevant interfaces are:
  - display and modify patient data,
  - manage prescriptions and the formulary database,
  - occupancy for housekeeping and maintenance, and
  - management level for aggregated data.

  Note that in some cases each interface represents a collection of two or more interfaces that are tailored to user needs.
- Some interfaces must be located in a sterile environment (e.g., operating rooms, emergency rooms, etc.). Their designs must be customized to suit these environments.
- The number of delivery units nation-wide is extremely large (up to 1 million?), since it must include all physicians, hospitals, insurance companies, medical product firms, etc. It is unclear whether such a system also extends into other accepted forms of medicine including chiropractic care and optometry. The volume of patient data is voluminous, especially if we mandate that all US citizens must have patient records. Issues related to centralization vs. distribution, replication, currency of information, number of interfaces with direct access, etc., must be considered!

Interfaces for HCA

The interfaces for HCA span a wide-spectrum of need and capabilities. To manage, control, access, and modify patient data, there are a number of interfaces required, with the variants determined by activities or responsibilities within a hospital:

- Nursing: This interface has moderate read and limited write capabilities on patient data. Writing is necessary to enter vital signs and other information as the treatment of the patient is implemented. This interface is further complicated since one would only want a specific nurse to be able to see data related to the patients that s/he is responsible for, and not all patients in a hospital.
- Physicians: This interface extends the one for Nursing, since MDs must be able to enter treatment orders/prescriptions.
- Administration: Nursing managers, department directors, and other administrators need a dedicated interface that provides aggregate and statistical results on the information in an HCA system.
- Pharmacy: This interface allows pharmacists to fill prescriptions per an MD's orders and also to check interactions of all drugs being given to a patient.
- Laboratory: This interface, as described in part in the previous section, would integrate the perform tests actions with the ability to automatically forward results when the test has been completed.

Interfaces for patients and their immediate family members may also be appropriate. In addition, there are interfaces required for the insurance companies that will process claims associated with patient treatment. Finally, interfaces related to occupancy and scheduling are necessary, to: admit patients for treatment, request housekeeping to preparing rooms upon a discharge, and file a maintenance request for repairs. To illustrate a portion of the interfaces and general operating environment, a partial DFD is given in Figure 3.2. The dashed line in the figure indicates that some of the required information may be located off-site.

Information for HCA

Given the DFD of Figure 3.2, it is possible to determine, at a high-level, the different databases that are required to engineer the information for HCA. For the various departments (e.g., pediatrics, nursery, surgical care, operating room, etc.) in a delivery unit (say, a hospital), the following databases must be available:

- Formulary: The collection of all drugs on the market that are used for treatment, their interactions with other medications, prescribing information, and possible side-effects.
- Prescriptions: The database of all prescriptions that have been filled by the Pharmacy per MD orders.
- Patient Records: Tracking all information on the patient's current visit and all past visits.
- Budgeting: Necessary database for the operational side of a hospital. Important to patient care, since costs related to equipment and personnel all factor in to the equation that determines treatment costs.

Figure 3.2: A Partial, Macro-Level DFD for HCA.

- Employee Records: A database that is maintained for all current and previous employees.
- Laboratory Tests/Results: A database that maintains information on all tests that have been ordered, their status, and their results.

These databases are local, since they exist within the delivery unit. However, there are mandatory links between these repositories and a collection of non-local databases, namely:

- MD Office: Contains patient records of visits at each MD's office.
- Medical Databank: A databank that might contain information on new and up-to-date treatments for various diseases.
- Drug Warnings/Updates: This database is necessary to report problems or recalls for various drugs.
- Insurance Companies: Databases that track the claims that have been processed for patients.

Clearly, the key issues regarding all of these databases involve currency and downloading (in both directions) for maintaining consistency and insuring confidentiality. In Figure 3.3, a partial ER diagram has been given for HCA, that provides a more detailed view for the above discussion.

Figure 3.3: A ER Diagram for HCA.

Security for HCA

In HCA, one goal of the security policy is to support the confidentiality of patient records. This is in direct conflict to the goal that requires reasonable and shared access to this same information. At a national-level, access to the patient records must be controlled. Within each delivery unit, access must be further restricted based on responsibilities. For example, the Nursing interface for HCA allows read access/limited writes on patients. However, this must be further refined to restrict access to those patients for whom a given Nurse (or other Health Care Professional) is directly responsible for. There are many units in a hospital, and in fact, VIPs, whose privacy must be protected. Clearly, confidentiality issues are pervasive throughout the health care delivery system. On the other hand, accessibility of patient records with identifying characteristics removed is also important. Such access allows statistical and medical trends to be queried and analyzed. For a certain rare disease, while a local overview is not possible (e.g., not enough patients), a national review of all patients with a given disease may allow researchers to identify trends and individual practitioners to predict health problems and establish preventive treatments.

At a finer-level of granularity, there are many different user roles and associated descriptions of responsibilities that are definable:

Nurse: *Direct involvement with patient care on a daily basis.*

Staff-RN: *Administer direct care to patients and implement the physician treatment plan.*

Education: *Educate both the nursing staff and patients regarding new treatments and self care.*

Manager: *Responsible for the day-to-day operation of a nursing unit.*

Vitals: *The actions taken to obtain the vital signs (pulse, BP, respiration) of patients.*

Once again, the responsibility level of the different user roles varies, ranging from the general (Nurse) to the specific (Vitals).

The security requirements for user roles are utilized to delineate the information access. Sample requirements for the three kinds of nurses given above are listed below:

- **Staff_RN:**
  - Clinical information on their own patients.
  - Write/modify a substantial portion of clinical information to record the results/patient progress.
  - Cannot change a Physician's orders on a patient.
- **Education:**
  - More restrictive access to clinical information than Staff_RN.
  - Need access to a patient's history to teach after-discharge care.
  - Can write notes which document a patient's progress.
  - Cannot change a Physician's orders on a patient.
- **Manager:**
  - Clinical information on all patients in their unit.
  - Additional information required to transfer patients between units, information on the nurses that work in their unit (including shifts, staffing, and skill levels), and budgetary data.
  - Write privileges of Staff_RN plus extra privileges to read information on other units (censuses)
  - Write summary and employee data on their own units.
  - Cannot change a Physician's orders on a patient.

Given these security requirements, it is possible to establish a ranking regarding the capabilities of each user role. From the above descriptions, Manager has more capabilities than Staff_RN which in turn has more capabilities than Education. This ranking is important, since if specified, it can be used to insure that a user role (say Education) is not given a privilege that is not present in a user role that is ranked above it (in this case, Staff_RN or Manager). Thus, privileges for different user roles can be analyzed and controlled.

References

[1] C. Ghezzi, M. Jazayeri, and D. Mandrioli, *Fundamentals of Software Engineering*, Prentice-Hall, 1991.

[2] M.-Y. Hu, S. Demurjian, and T.C. Ting, "User-Role Based Security Profiles for an Object-Oriented Design Model", in *Database Security, VI: Status and Prospects*, C. Landwehr and B. Thuraisingham (eds.), North-Holland, 1993.

[3] T.C. Ting, S. Demurjian, and M.-Y. Hu, "A Specification Methodology for User-Role Based Security in an Object-Oriented Design Model - Experience with a Health Care Application", *Proc. of Sixth IFIP WG11.3 Working Conf. on Database Security*, Vancouver, Canada, Aug. 1992.

[4] D. Tsichritzis and F. Lochovsky, *Data Models*, Prentice-Hall, 1982.