K.D. K. College of Engineering

Department of Information Technology

B.E. (Information Technology) Fifth Semester (C.B.S.)

Software Engineering

UNIT: 03

System Engineering

Systems engineering (SE) is an interdisciplinary area of endeavor whose focus is the development of complex techonolgical systems with reference to their extended environment. Software engineering encompasses the design, development and maintenance of complex systems with consideration to their software and hardware, their interconnections and the environments in which they operate over the course of their life cycle and ultimate decommissioning.

Systems involve multiple elements that are organized for a given purpose; complex systems are, by definition, multifaceted, intricately connected, which makes them challenging to plan for and maintain. Most technological systems can be more effectively be thought of as systems of systems (SoS) to direct the focus to how constituent parts interoperate, work over time and function within the context of a larger, evolving system.

Systems thinking, in general, is a holistic approach that focuses on the way that a system's constituent parts interrelate and how systems work over time and within the context of larger systems. Current and developing trends of increasing automation and the Internet of Things (IoT) are necessitating a more holistic view of the increasingly complex and interconnected systems operating throughout business and industry (*See:* IIoT).

SE integrates knowledge and methods from multiple fields of science and technology to spark innovation in the development and implementation of new technologies. Disciplines informing SE include control engineering, cybernetics, electrical engineering, engineering management, industrial engineering, manufacturing engineering, mechanical engineering, organizational studies, project management and software engineering.

THE SYSTEM ENGINEERING HIERARCHY

System Engineering encompasses a collection of top- down and methods to navigate the hierarchy illustrated in figure 3.1. The system engineering process usually begins with a “world view”. That is, the entire business

or product domain is examined to ensure that the proper business or technology context can be established. The world view is refined to focus more fully on a specific domain of interest. Within a specific domain, the need for targeted system elements is analysed. Finally, the analysis, design, and construction of a targeted system elements are initiated. At the top of the hierarchy, a very broad context is established and, at the bottom, detailed activities, performed by the relevant engineering disciplines are conducted.

The world view (WV) is composed of a set of domains (D1), which can each be a system or system of systems in its own right

WV = {D1 D2 D3……,Dn }

Each domain is composed of specific elements (Ej) each of which serves some role in accomplishing the objective and goals of the domain and component:

Di = {E1 E2 E3 …… Em}

Finally, each element is implemented by specifying the technical components (Ck) that achieve the necessary function for an element:

Ej = {C1, C2 C3 …….. C k}

In the software context, a component could be a computer program, a reusable program component, a module, a class or object, or even a programming language statement.

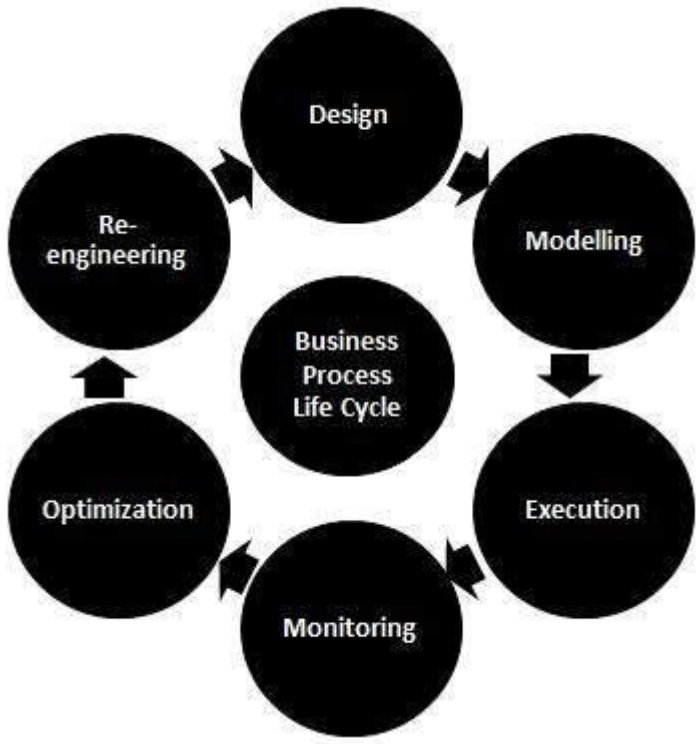
Business Process Engineering

A business process is a collection of activities or tasks that produce a specific service or product for end users. It is usually represented as a flowchart as a sequence of activities that points to a Process Matrix.

Business process Modelling is carried out by Process owners or product owners to enable the test team to test efficiently. It aims to improve business performance by optimising the efficiency of the associated activities of a product or service.

Business Process Life Cycle:

Business Process Life cycle has various phases as listed below:



Business process engineering is a way in which organizations study their current business processes and develop new methods to improve productivity, efficiency, and operational costs. Business process engineering focuses on new business processes, how to diagnose problems with an organization's current methodology, and how to redesign, reconstruct, and monitor processes to ensure they are effective.[1]

Business Process Engineering (BPE) uses a proven systematic approach based on the latest experiences and research to achieve significant improvements. The BPE process helps clients fundamentally rethink and reinvent the business processes needed to achieve the firm's strategic objectives through the maximum use of enabling technologies and organizational strategies. A BPE effort can result in 15% to 50% improvement in performance of the targeted business processes, depending upon whether a reengineering or an improvement approach is used in the effort.[2]

Business Process Engineering Approach[3]

Business Process Engineering (BPE) Approach includes:

- Understanding the Present Mode of Operation (PMO). We'll assemble an experienced team to analyze your current processes, technologies and systems. The result will be creation of a detailed PMO business process model showing interrelationships and dependencies between people, systems, and processes. This will serve as the baseline that proposed changes and actual implementations are evaluated against.
- Determining the Future Mode of Operation (FMO). We'll work with you to build an advisory team to define an FMO business process model based on your business objectives and our combined knowledge of industry best practices.
- Gap Analysis and Transition Plan. A gap analysis of needed business process improvements and transition to the FMO plan will be developed. You'll gain an understanding of the business strategy, timing, personnel, and system/process evolution that will take place.

- Implementation. By trialing and deploying new system or operational process improvements, we'll help you determine whether the intended results will be achieved. If additional system and operational process improvements need to be made, we'll repeat the appropriate PMO, FMO, Gap Analysis and Transition Planning steps as necessary.

Conceptual Framework for Business Process Engineering (See Figure 1.)[4] The following is a proposed framework within the requirements engineering discipline. It covers business process capture and modeling, elicitation of business and technical requirements for the application and the proposal of the process changes requested for satisfactory application implementation. The framework was originally developed for a particular case study, but is not strictly limited to its domain. Figure 1 shows the essential process elements of the framework.

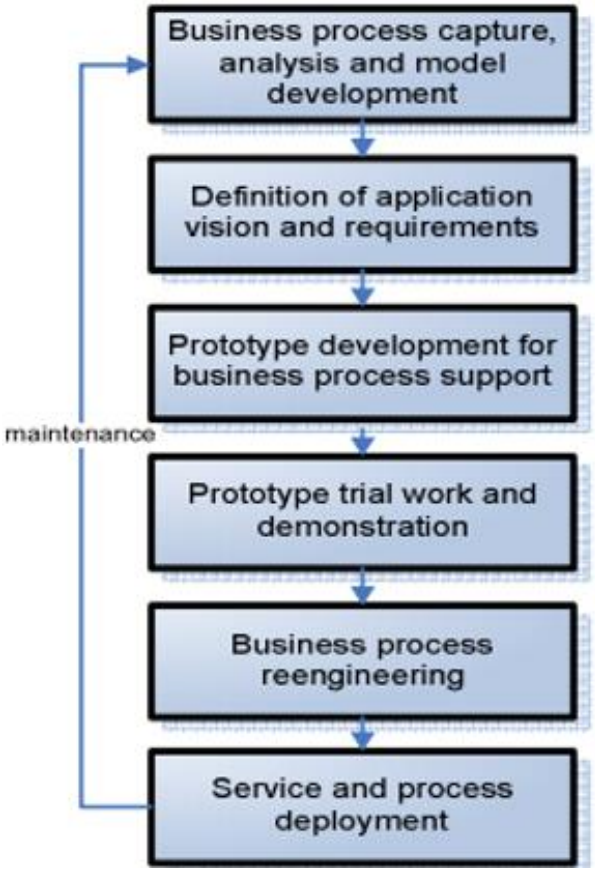


Figure 1. source: University of Zagreb

- Business process capture, analysis and model development: Business process capture, analysis and model development include the following. The initial boundaries of the business process are identified and a basic understanding of the process is acquired. The process is then described and modeled. The process actors are analyzed and service user groups are identified. The activities performed by each user group are recognized and modeled. The requirements elicitation techniques used in this stage are observations, interviews and brainstorming sessions.
- Definition of application vision and requirements. To define the vision of an application and its requirements the following is done. Stakeholders' requirements are gathered and mapped into service features to produce high-level functional specifications. Questionnaires are prepared and distributed to

the identified user groups. This activity should be based on a clear vision and a promising business model. Any discussion with authorities should be initiated here.

- Prototype development for business process support. A service prototype is designed and implemented according to the requirements specification. The prototype aims at gathering and verifying complete user requirements and provides more concrete features such as a notification service.
- Prototype trial work and demonstration. Trial work is not only important for application functionality and performance verification but for the school process as well. Process weaknesses are emerging, so proposals for process changes must be explicitly defined and clearly presented to all process actors. The regularized rules must contain original process description and functional requirements of the new service and their implications on the business process. Also, each process change must be unambiguously stated and illustrated. The new process model must be easily understood by the users.
- Business process reengineering. The only way to propose acceptable process changes is by collecting proposals directly from the process actors. Their on-topic feedback is valuable from both a technological and social aspect. Process actors can be regarded as domain experts in their field. Consequently, requirements specifications should be refined and users'feedback about process changes collected. Appropriate techniques include questionnaires, interviews, small group discussions and specialized workshops.
- Service and process deployment. Finally, the service and improved process are deployed and set into motion. First step is mobilizing all the actors involved. The requirements are subject to change and user groups' representatives should be involved in all the development iterations, particularly deployment. Adequate user training should be ensured and the support of service experts is highly recommended. These are the key success factors for this phase.Regular presentations of the development activities and control of the changing requirements are aimed at ensuring service quality. Any change which influences the process is promptly introduced into the new process model and discussed with the actors with the possibility of being implemented in the next cycle.

Business Process Testing [BPT]:

It is a tool used for an automated and manual testing for designing tests, maintaining tests and executing tests. The reusable tests are usually designed by Business Analysts for improving test efficiency.

Benefits and Features of BPT :

- Allows non-technical subject matter expertise to quickly build a reusable test workflow.
- It reduces the effort required for test maintenance.
- It converts the manual tests to manual test components.
- It provides a framework to build User Acceptance Tests to meet the requirements.

Product Engineering

Software Products are nothing but software systems delivered to the customer with the documentation that that describe how to install and use the system. In certain cases, software products may be part of system products where hardware, as well as software, is delivered to a customer. Software products are produced with the help of the software process. The software process is a way in which we produce software.
Types of software products:
Software products fall into two broad categories:

1. Generic products:
   Generic products are the stand-alone systems that are developed by a production unit and sold on the open market to any customer who is able to buy them.

2. Customised Products:
   Customised products are the systems that are commissioned by a particular customer. Some contractor develops the software for that customer.

Essential characteristics of Well-Engineered Software Product:
A well-engineered software product should possess the following essential characteristics:

- Efficiency:
  The software should not make wasteful use of system resources such as memory and processor cycles.
- Maintainability:
  It should be possible to evolve the software to meet the changing requirements of customers.
- Dependability:
  It is the flexibility of the software that ought to not cause any physical or economic injury within the event of system failure. It includes a range of characteristics such as reliability, security and safety.
- In time:
  Software should be developed well in time.
- Within Budget:
  The software development costs should not overrun and it should be within the budgetary limit.
- Functionality:
  The software syatem should exhibit the proper functionality, i.e, it should perform all the functions it is supposed to perform.
- Adaptability:
  The software system should have the ability to get adapted to a reasonable extent with the changing requirements.

- **Product engineering** refers to the process of designing and developing a device, assembly, or system such that it be produced as an item for sale through some production manufacturing process. Product engineering usually entails activity dealing with issues of cost, producibility, quality, performance, reliability, serviceability, intended lifespan and user features. These product characteristics are generally all sought in the attempt to make the resulting product attractive to its intended market and a successful contributor to the business of the organization that intends to offer the product to that market. It includes design, development and transitioning to manufacturing of the product. The term encompasses developing the concept of the product and the design and development of its mechanical, electronics and software components. After the initial design and development is done, transitioning the product to manufacture it in volumes is considered part of product engineering.
- For example, the engineering of a digital camera would include defining the feature set, design of the optics, the mechanical and ergonomic design of the packaging, developing the electronics that control the various component and developing the software that allows the user to see the pictures, store them in memory and download them to a computer.
- Product engineering is an engineering discipline that deals with both design and manufacturing aspects of a product.

Area of responsibility[edit]

- Product engineers define the yield road map and drive the fulfillment during ramp-up and volume production,
- Identify and realize measures for yield improvement, test optimization and product cost-ability methods,
- Define qualification plan and perform feasibility analysis.

Product engineers are the technical interface between the component development team and the production side (Front End and Back End), especially after the development phase and qualifications when the high volume production is running.

Product engineers improve the product quality and secure the product reliability by balancing the cost of tests and tests coverage that could impact the production fall-off. They support failure analysis request from customers.

System Modeling

System modeling is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system. It is about representing a system using some kind of graphical notation, which is now almost always based on notations in the Unified Modeling Language (UML). Models help the analyst to understand the functionality of the system; they are used to communicate with customers.

Models can explain the system from different perspectives:

- An external perspective, where you model the context or environment of the system.
- An interaction perspective, where you model the interactions between a system and its environment, or between the components of a system.
- A structural perspective, where you model the organization of a system or the structure of the data that is processed by the system.
- A behavioral perspective, where you model the dynamic behavior of the system and how it responds to events.

Five types of UML diagrams that are the most useful for system modeling:

- Activity diagrams, which show the activities involved in a process or in data processing.
- Use case diagrams, which show the interactions between a system and its environment.
- Sequence diagrams, which show interactions between actors and the system and between system components.
- Class diagrams, which show the object classes in the system and the associations between these classes.
- State diagrams, which show how the system reacts to internal and external events.

Models of both new and existing system are used during requirements engineering. Models of the existing systems help clarify what the existing system does and can be used as a basis for discussing its strengths and weaknesses. These then lead to requirements for the new system. Models of the new system are used during requirements engineering to help explain the proposed requirements to other system stakeholders. Engineers use these models to discuss design proposals and to document the system for implementation.

Context and process models

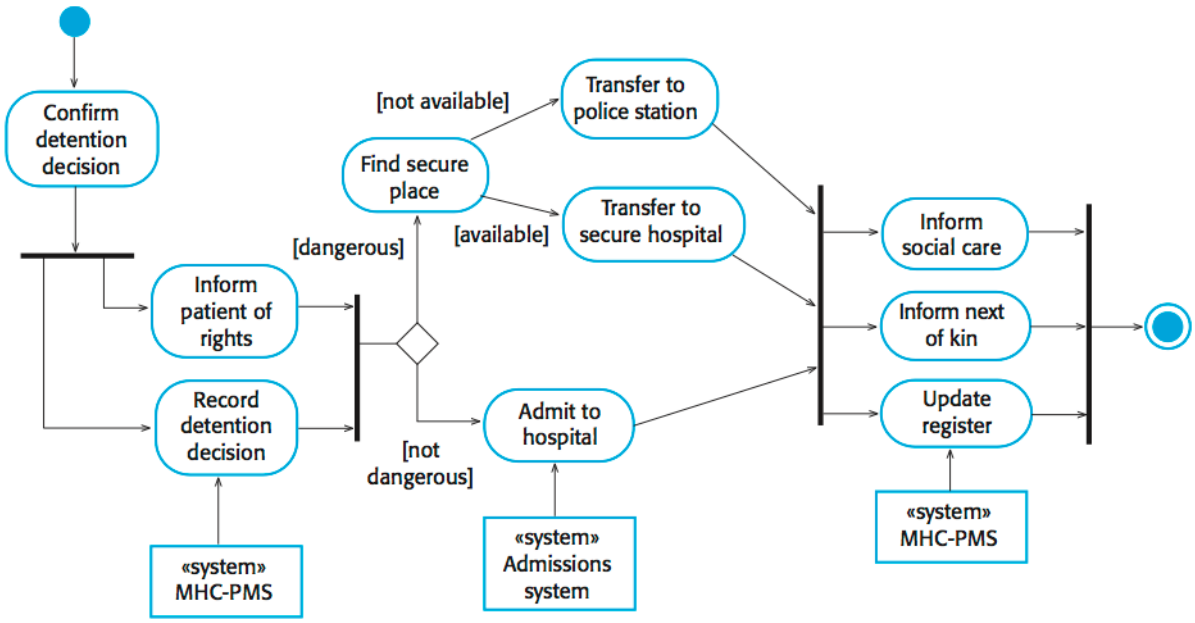
Context models are used to illustrate the operational context of a system - they show what lies outside the system boundaries. Social and organizational concerns may affect the decision on where to position system boundaries. Architectural models show the system and its relationship with other systems.

System boundaries are established to define what is inside and what is outside the system. They show other systems that are used or depend on the system being developed. The position of the system boundary has a profound effect on the system requirements. Defining a system boundary is

a political judgment since there may be pressures to develop system boundaries that increase/decrease the influence or workload of different parts of an organization.

Context models simply show the other systems in the environment, not how the system being developed is used in that environment. Process models reveal how the system being developed is used in broader business processes. UML activity diagrams may be used to define business process models.

The example below shows a UML activity diagram describing the process of involuntary detention and the role of MHC-PMS (mental healthcare patient management system) in it.



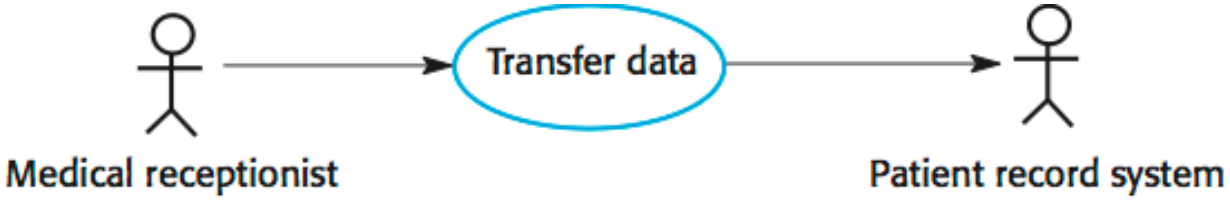
Interaction models

Types of interactions that can be represented in a model:

- Modeling user interaction is important as it helps to identify user requirements.
- Modeling system-to-system interaction highlights the communication problems that may arise.
- Modeling component interaction helps us understand if a proposed system structure is likely to deliver the required system performance and dependability.

Use cases were developed originally to support requirements elicitation and now incorporated into the UML. Each use case represents a discrete task that involves external interaction with a system. Actors in a use case may be people or other systems. Use cases can be represented using a UML use case diagram and in a more detailed textual/tabular format.

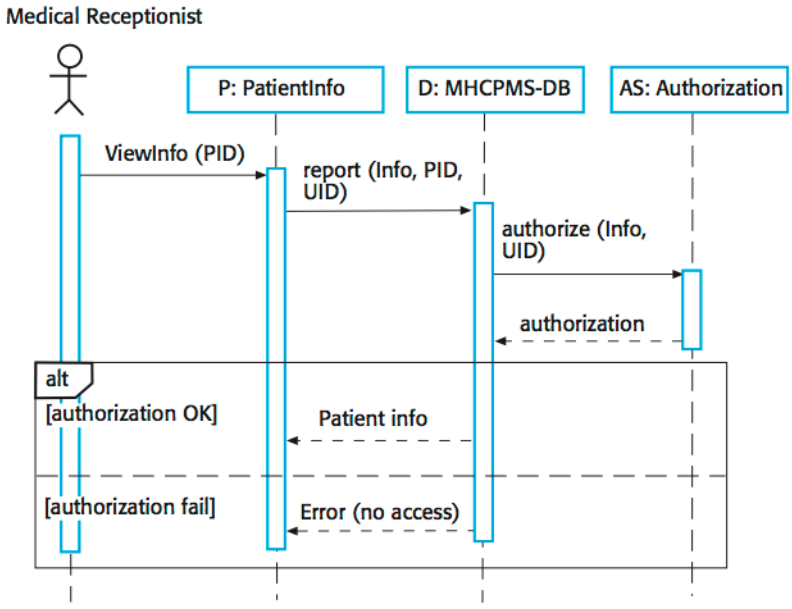
Simple use case diagram:



Use case description in a tabular format:

Use case title	Transfer data
Description	A receptionist may transfer data from the MHC-PMS to a general patient record database that is maintained by a health authority. The information transferred may either be updated personal information (address, phone number, etc.) or a summary of the patient's diagnosis and treatment.
Actor(s)	Medical receptionist, patient records system (PRS)
Preconditions	Patient data has been collected (personal information, treatment summary); The receptionist must have appropriate security permissions to access the patient information and the PRS.
Postconditions	PRS has been updated
Main success scenario	1. Receptionist selects the "Transfer data" option from the menu. 2. PRS verifies the security credentials of the receptionist. 3. Data is transferred. 4. PRS has been updated.
Extensions	2a. The receptionist does not have the necessary security credentials. 2a.1. An error message is displayed. 2a.2. The receptionist backs out of the use case.

UML sequence diagrams are used to model the interactions between the actors and the objects within a system. A sequence diagram shows the sequence of interactions that take place during a particular use case or use case instance. The objects and actors involved are listed along the top of the diagram, with a dotted line drawn vertically from these. Interactions between objects are indicated by annotated arrows.

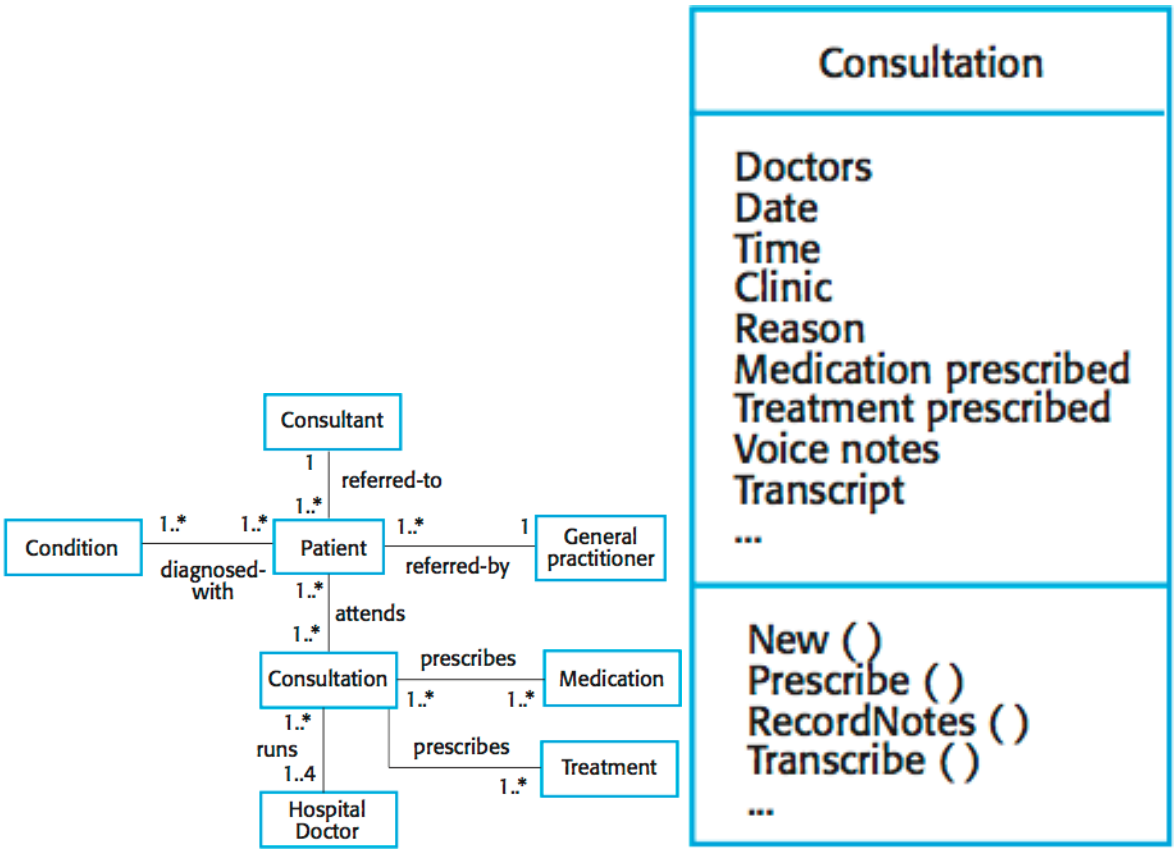


Structural models

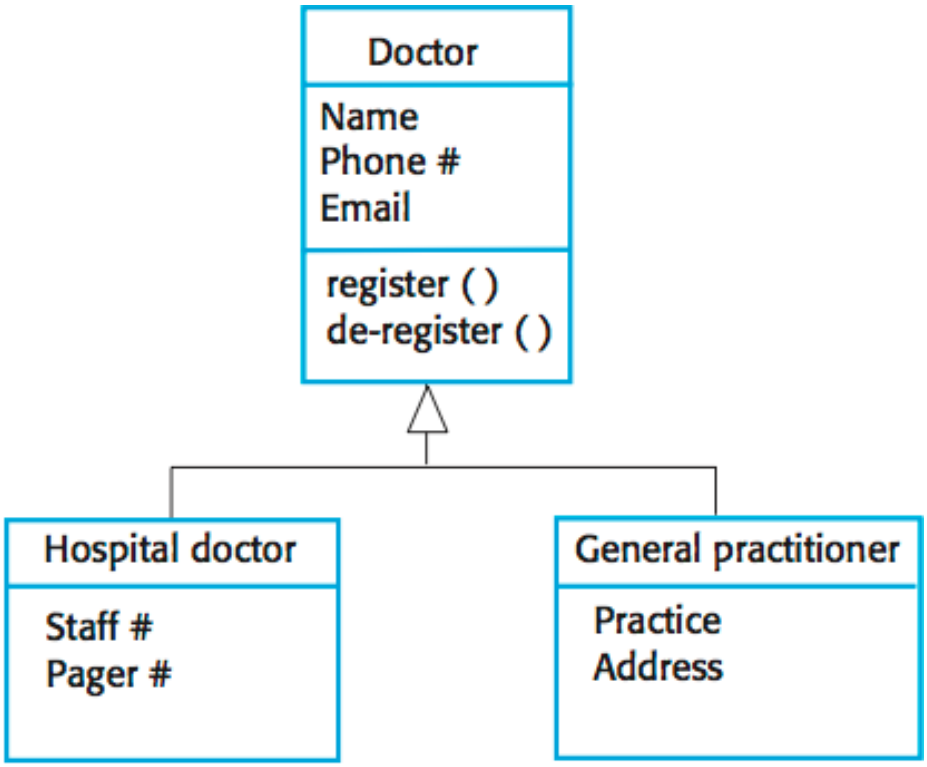
Structural models of software display the organization of a system in terms of the components that make up that system and their relationships. Structural models may be static models, which show

the structure of the system design, or dynamic models, which show the organization of the system when it is executing. You create structural models of a system when you are discussing and designing the system architecture.

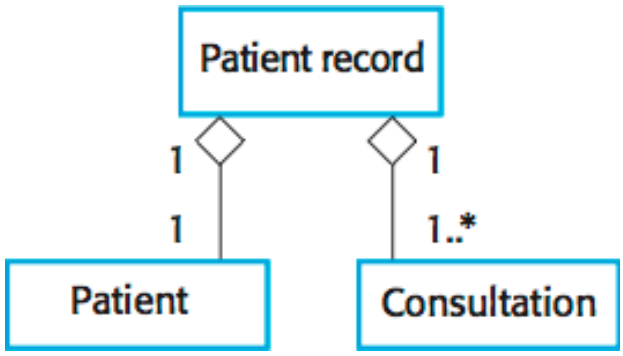
UML class diagrams are used when developing an object-oriented system model to show the classes in a system and the associations between these classes. An object class can be thought of as a general definition of one kind of system object. An association is a link between classes that indicates that there is some relationship between these classes. When you are developing models during the early stages of the software engineering process, objects represent something in the real world, such as a patient, a prescription, doctor, etc.



Generalization is an everyday technique that we use to manage complexity. In modeling systems, it is often useful to examine the classes in a system to see if there is scope for generalization. In object-oriented languages, such as Java, generalization is implemented using the class inheritance mechanisms built into the language. In a generalization, the attributes and operations associated with higher-level classes are also associated with the lower-level classes. The lower-level classes are subclasses inherit the attributes and operations from their superclasses. These lower-level classes then add more specific attributes and operations.



An aggregation model shows how classes that are collections are composed of other classes. Aggregation models are similar to the part-of relationship in semantic data models.

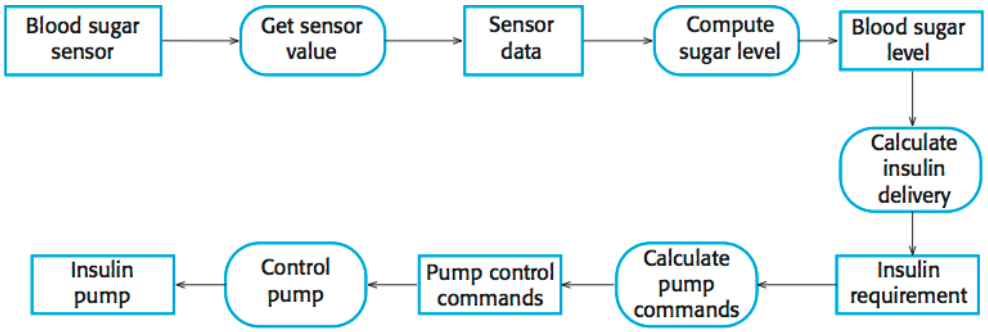


Behavioral models

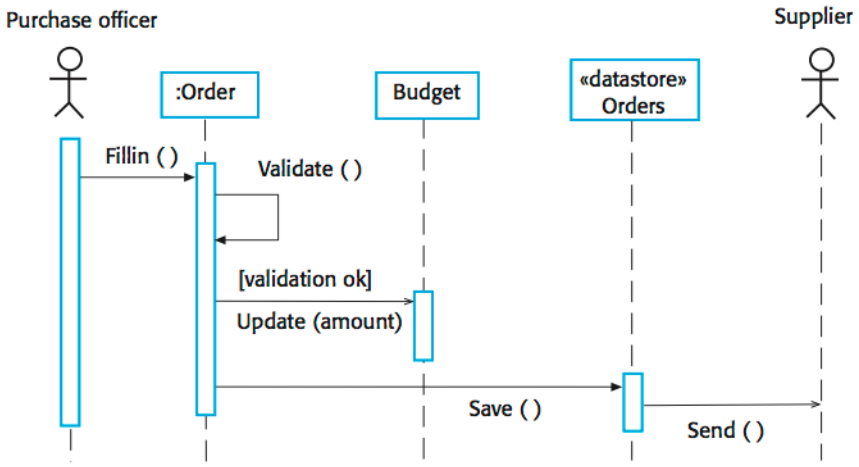
Behavioral models are models of the dynamic behavior of a system as it is executing. They show what happens or what is supposed to happen when a system responds to a stimulus from its environment. Two types of stimuli:

- Some data arrives that has to be processed by the system.
- Some event happens that triggers system processing. Events may have associated data, although this is not always the case.

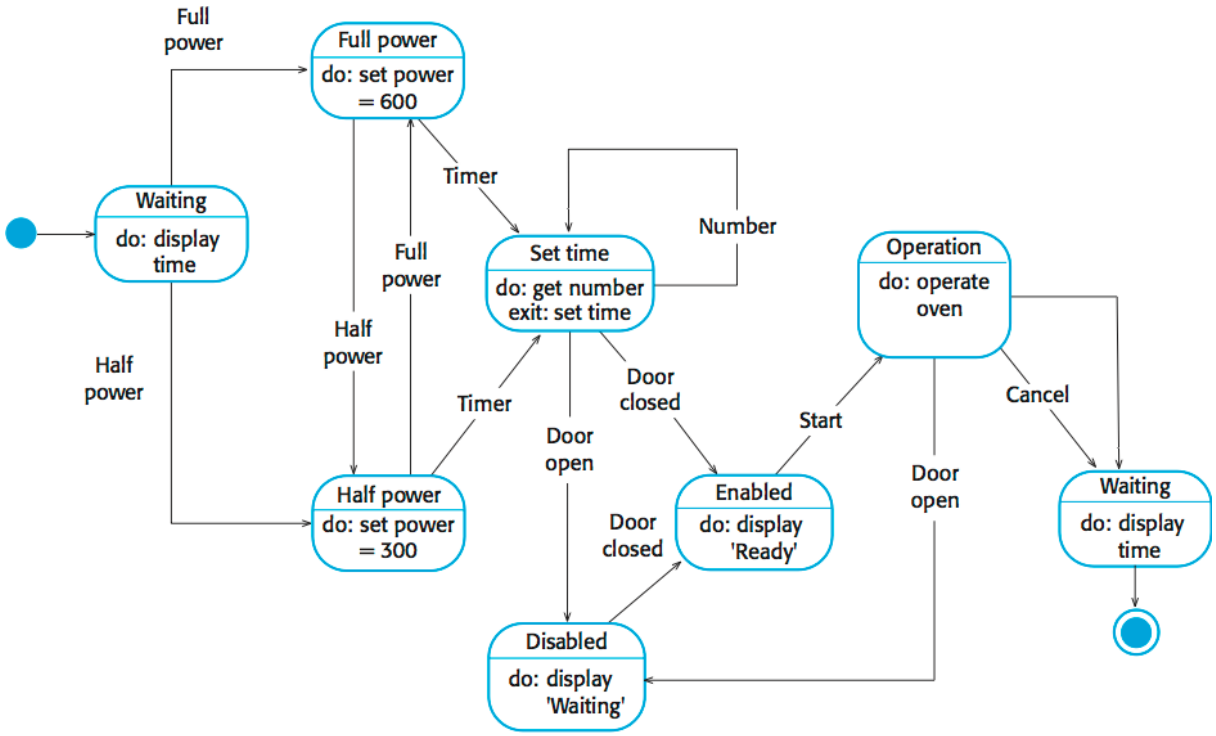
Many business systems are data-processing systems that are primarily driven by data. They are controlled by the data input to the system, with relatively little external event processing. Data-driven models show the sequence of actions involved in processing input data and generating an associated output. They are particularly useful during the analysis of requirements as they can be used to show end-to-end processing in a system. Data-driven models can be created using UML activity diagrams:



Data-driven models can also be created using UML sequence diagrams:



Real-time systems are often event-driven, with minimal data processing. For example, a landline phone switching system responds to events such as 'receiver off hook' by generating a dial tone. Event-driven models shows how a system responds to external and internal events. It is based on the assumption that a system has a finite number of states and that events (stimuli) may cause a transition from one state to another. Event-driven models can be created using UML state diagrams:



Software Requirements Engineering

Introduction to requirement engineering

- The process of collecting the software requirement from the client then understand, evaluate and document it is called as requirement engineering.
- Requirement engineering constructs a bridge for design and construction.

Requirement engineering consists of seven different tasks as follow:

1. Inception

- Inception is a task where the requirement engineering asks a set of questions to establish a software process.
- In this task, it understands the problem and evaluates with the proper solution.
- It collaborates with the relationship between the customer and the developer.
- The developer and customer decide the overall scope and the nature of the question.

2. Elicitation

Elicitation means to find the requirements from anybody. The requirements are difficult because the following problems occur in elicitation.

Problem of scope: The customer give the unnecessary technical detail rather than clarity of the overall system objective.

Problem of understanding: Poor understanding between the customer and the developer regarding various aspect of the project like capability, limitation of the computing environment.

Problem of volatility: In this problem, the requirements change from time to time and it is difficult while developing the project.

3. Elaboration

- In this task, the information taken from user during inception and elaboration and are expanded and refined in elaboration.
- Its main task is developing pure model of software using functions, feature and constraints of a software.

4. Negotiation

- In negotiation task, a software engineer decides the how will the project be achieved with limited business resources.
- To create rough guesses of development and access the impact of the requirement on the project cost and delivery time.

5. Specification

- In this task, the requirement engineer constructs a final work product.
- The work product is in the form of software requirement specification.
- In this task, formalize the requirement of the proposed software such as informative, functional and behavioral.
- The requirement are formalize in both graphical and textual formats.

6. Validation

- The work product is built as an output of the requirement engineering and that is accessed for the quality through a validation step.
- The formal technical reviews from the software engineer, customer and other stakeholders helps for the primary requirements validation mechanism.

7. Requirement management

- It is a set of activities that help the project team to identify, control and track the requirements and changes can be made to the requirements at any time of the ongoing project.
- These tasks start with the identification and assign a unique identifier to each of the requirement.
- After finalizing the requirement traceability table is developed.
- The examples of traceability table are the features, sources, dependencies, subsystems and interface of the requirement.

Eliciting Requirements

Eliciting requirement helps the user for collecting the requirement

Eliciting requirement steps are as follows:

1. Collaborative requirements gathering

- Gathering the requirements by conducting the meetings between developer and customer.
- Fix the rules for preparation and participation.
- The main motive is to identify the problem, give the solutions for the elements, negotiate the different approaches and specify the primary set of solution requirements in an environment which is valuable for achieving goal.

2. Quality Function Deployment (QFD)

- In this technique, translate the customer need into the technical requirement for the software.
- QFD system designs a software according to the demands of the customer.

QFD consist of three types of requirement:

Normal requirements

- The objective and goal are stated for the system through the meetings with the customer.
- For the customer satisfaction these requirements should be there.

Expected requirement

- These requirements are implicit.
- These are the basic requirement that not be clearly told by the customer, but also the customer expect that requirement.

Exciting requirements

- These features are beyond the expectation of the customer.
- The developer adds some additional features or unexpected feature into the software to make the customer more satisfied.

For example, the mobile phone with standard features, but the developer adds few additional functionalities like voice searching, multi-touch screen etc. then the customer more exited about that feature.

3. Usage scenarios

- Till the software team does not understand how the features and function are used by the end users it is difficult to move technical activities.
- To achieve above problem the software team produces a set of structure that identify the usage for the software.
- This structure is called as 'Use Cases'.

4. Elicitation work product

- The work product created as a result of requirement elicitation that is depending on the size of the system or product to be built.
- The work product consists of a statement need, feasibility, statement scope for the system.
- It also consists of a list of users participate in the requirement elicitation.

Analysis Model in Software Engineering

- Analysis model operates as a link between the 'system description' and the 'design model'.
- In the analysis model, information, functions and the behaviour of the system is defined and these are translated into the architecture, interface and component level design in the 'design modeling'.

Elements of the analysis model

1. Scenario based element

- This type of element represents the system user point of view.
- Scenario based elements are use case diagram, user stories.

2. Class based elements

- The object of this type of element manipulated by the system.
- It defines the object,attributes and relationship.
- The collaboration is occurring between the classes.
- Class based elements are the class diagram, collaboration diagram.

3. Behavioral elements

- Behavioral elements represent state of the system and how it is changed by the external events.
- The behavioral elements are sequenced diagram, state diagram.

4. Flow oriented elements

- An information flows through a computer-based system it gets transformed.
- It shows how the data objects are transformed while they flow between the various system functions.
- The flow elements are data flow diagram, control flow diagram.

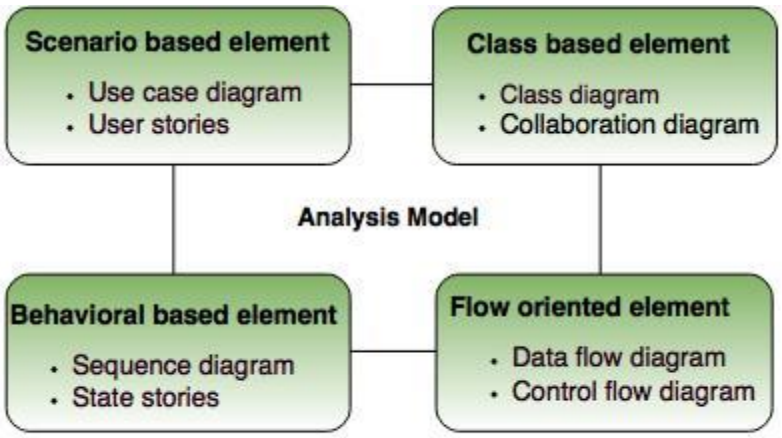


Fig. - Elements of analysis model

Analysis Rules of Thumb

The rules of thumb that must be followed while creating the analysis model.

The rules are as follows:

- The model focuses on the requirements in the business domain. The level of abstraction must be high i.e there is no need to give details.
- Every element in the model helps in understanding the software requirement and focus on the information, function and behaviour of the system.
- The consideration of infrastructure and nonfunctional model delayed in the design. For example, the database is required for a system, but the classes, functions and behavior of the database are not initially required. If these are initially considered then there is a delay in the designing.
- Throughout the system minimum coupling is required. The interconnections between the modules is known as 'coupling'.
- The analysis model gives value to all the people related to model.
- The model should be simple as possible. Because simple model always helps in easy understanding of the requirement.

Concepts of data modeling

- Analysis modeling starts with the data modeling.
- The software engineer defines all the data object that proceeds within the system and the relationship between data objects are identified.

Data objects

- The data object is the representation of composite information.
- The composite information means an object has a number of different properties or attribute. For example, Height is a single value so it is not a valid data object, but dimensions contain the height, the width and depth these are defined as an object.

Data Attributes
Each of the data object has a set of attributes.

Data object has the following characteristics:

- Name an instance of the data object.
- Describe the instance.
- Make reference to another instance in another table.

Relationship
Relationship shows the relationship between data objects and how they are related to each other.

Cardinality
Cardinality state the number of events of one object related to the number of events of another object.

The cardinality expressed as:

One to one (1:1)
One event of an object is related to one event of another object.
For example, one employee has only one ID.

One to many (1:N)
One event of an object is related to many events.
For example, One collage has many departments.

Many to many(M:N)
Many events of one object are related to many events of another object.
For example, many customer place order for many products.

Modality

- If an event relationship is an optional then the modality of relationship is zero.
- If an event of relationship is compulsory then modality of relationship is one.

Scenario-Based Requirements Modeling:

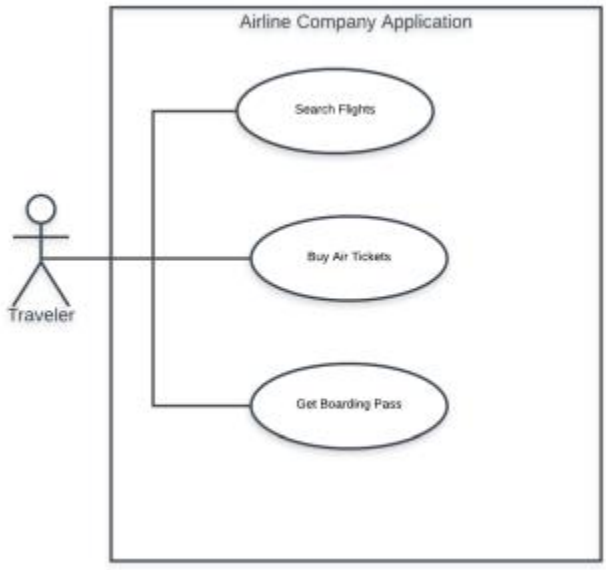
To understand scenario-based modeling, you first need to understand requirements modeling and how it applies to software engineering in general. Requirements modeling is essentially the planning stage of a software system or application. The path by which such a system or application comes to life will generally begin with a business or entity identifying a problem they're facing that requires a software solution, and they approach a software development team to come up with this solution. Requirements modeling is the process of identifying the requirements this software solution must meet in order to be successful. Requirements modeling contains several sub-stages, typically: scenario-based modeling, flow-oriented modeling, data modeling, class-based modeling and behavioral modeling. Also, as the term 'modeling' implies, all of these stages typically result in producing diagrams that visually convey the concepts they identify. The most common method for creating these diagrams is UML, Unified Modeling Language.

What Does Scenario-Based Modeling Do?

While technically there is no 'right way' to go through the stages of requirements modeling, it typically begins with scenario-based modeling, and that is because it identifies the possible use cases for the system and produces the use case diagram, to which all the other stages of requirements modeling refer.

Use Case

The use case is essentially a primary example of how the proposed software application or system is meant to be used, from the user's point of view. A use case diagram will typically show system 'actors' (humans or other entities external to the system) and how they interact with the system. Technically, each action such a system actor can perform with the application or system is considered to be a separate use case.



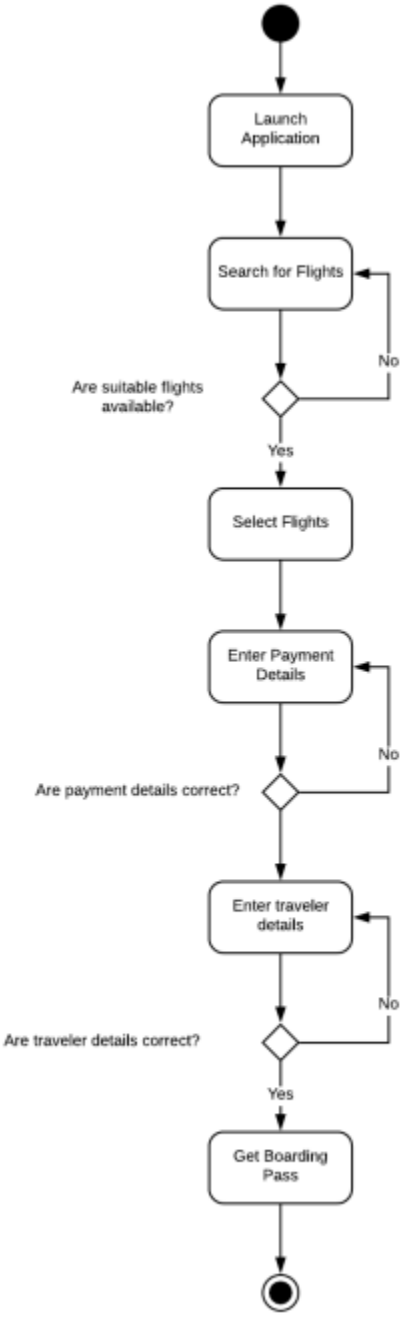
The example use case diagram above, drawn using UML, depicts three possible use cases for the system actor 'Traveler' when interacting with an airline company's application. The three use cases / actions depicted are:

1. Search for flights
2. Buy tickets
3. Get boarding pass

The large rectangle labeled 'Airline Company Application' represents the system boundary. Notice that the actor labeled 'Traveler' is outside the system boundary. In UML, all actors are depicted outside the system boundary.

Activity Diagram

In addition to the use case diagram, scenario-based modeling can also produce an activity diagram, which is essentially a more advanced flow chart that depicts how one activity leads to another while using a system.



Object Oriented Analysis (OOA)

Object Oriented (OO) techniques can be applied in the phases of software life cycle (analysis, design, implementation, etc). This article describes about *object oriented analysis, how to use OOA in software development*? Summary of the article:

- What is Structural Analysis?
- What is Object Oriented Analysis (OOA)?
- Functionalities' of OOA
- Advantages of OOA
- Structured Analysis VS Object Oriented Analysis

What is Structural Analysis?

The Structural Analysis/Structured Design (SASD) approach is the traditional approach of software development. It is based upon the waterfall model. The development phases of a SASD system are given bellow:

- Feasibility Study
- Requirement Analysis and Specification
- System Design
- Implementation
- Post-implementation Review

What is Object Oriented Analysis (OOA)?

Object Oriented Analysis (OOA) is process of discovery where a development team understands and models the requirements of the system. In OOA requirements are organized as objects. It integrates all the process and data. But in others or traditional structural analysis both process and data are considered independently/separately. They use flow chart/structure charts for process and ER diagrams for data.

But In OOA some advance models are used. The common models used in OOA are: Use cases, Object models. Use cases describe pictures or overview for standard domain functions that the system must achieved. Object models describe the names, class relations, operations, and properties of the main objects. User-interface prototypes can also be created for better understanding.

Object Oriented Analysis (OOA) begins by looking at the problem domain (the area of expertise or application that needs to analyze in order to solve a problem). Its aim is to produce a conceptual model of the information that exists in the area being analyzed. For the analysis there are a variety of sources. It can be a formal document, a written requirements statement, interviews with stakeholders/other interested parties, other methods, etc. The final result of object oriented analysis will appear in the form of a conceptual model that describes what the system is functionally required to do.

Functionalities of OOA

The core activities in OOA are given bellow:

- Find the objects
- Organize the objects by creating object model diagram
- Explain how the objects communicates with each others
- Set the characteristic or behavior of the objects
- Set the internal of the objects

Advantages of OOA

The OOA provides better performance. Some common advantages of OOA are given bellow:

- Its focuses on data rather than the procedures as in Structured Analysis
- The objectives of encapsulation and data hiding help the developer to develop the systems that cannot be tampered by other parts of the system
- It allows effective software complexity management by the virtue of modularity
- It can be upgraded from small to large system easily

Structured Analysis VS Object Oriented Analysis

Some differences between Structured Analysis and Object Oriented Analysis are given bellow:

- Structured Analysis treats processes and data as separate components. Where, OO Analysis Combines processes and data into single component which called object
- Structured Analysis does not support re-usability of code. So, the development time and cost is naturally high. But OO analysis supports code re-usability which reduce the development time and cost

The Object Oriented Analysis and Design (OOAD) is a new dimension to develop the software system. It provides better performance comparatively to tradition structured analysis. But in some case it is inappropriate. We should consider the advantages and disadvantages of both OOA and structured analysis to choice correct techniques.

Data Modeling

Data modeling is the process of creating a data model for the data to be stored in a Database. This data model is a conceptual representation of

- Data objects
- The associations between different data objects
- The rules.

Data modeling helps in the visual representation of data and enforces business rules, regulatory compliances, and government policies on the data. Data Models ensure consistency in naming conventions, default values, semantics, security while ensuring quality of the data.

Data model emphasizes on what data is needed and how it should be organized instead of what operations need to be performed on the data. Data Model is like architect's building plan which helps to build a conceptual model and set the relationship between data items.

The two types of Data Models techniques are

1. Entity Relationship (E-R) Model
2. UML (Unified Modelling Language)

We will discuss them in detail later.

In this tutorial, you will learn-

- What is Data Modelling?
- Why use Data Model?
- Types of Data Models
  - Conceptual Model
  - Logical Data Model
  - Physical Data Model
- Advantages and Disadvantages of Data Model

Why use Data Model?

The primary goal of using data model are:

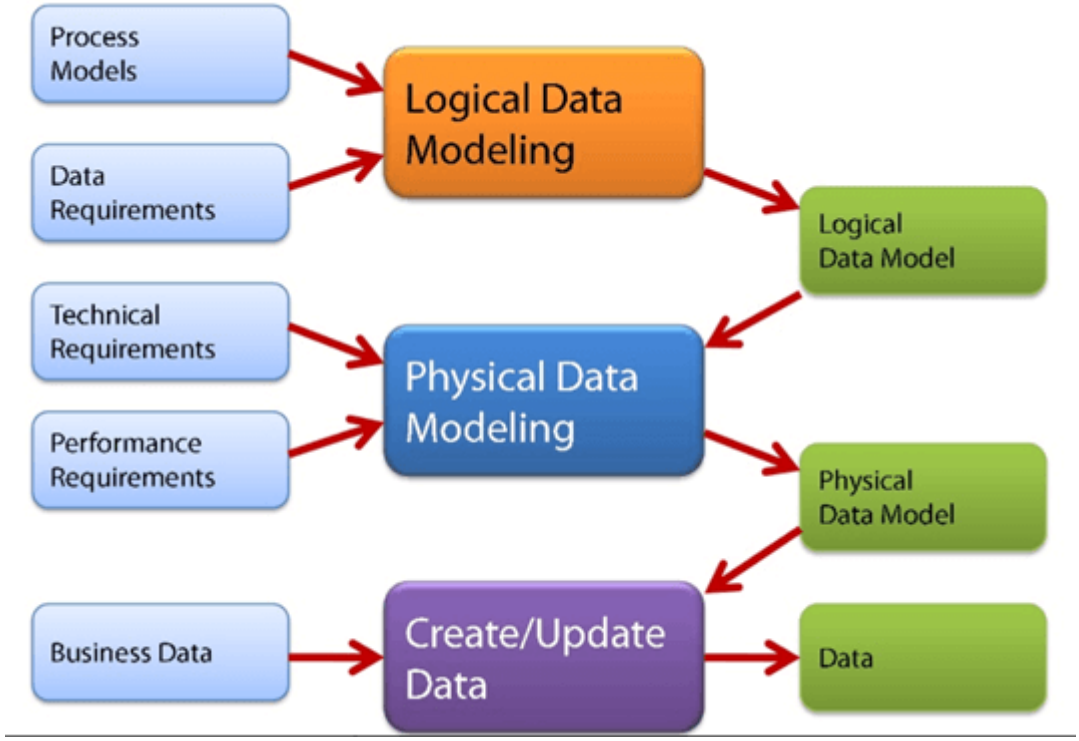
- Ensures that all data objects required by the database are accurately represented. Omission of data will lead to creation of faulty reports and produce incorrect results.
- A data model helps design the database at the conceptual, physical and logical levels.
- Data Model structure helps to define the relational tables, primary and foreign keys and stored procedures.

- It provides a clear picture of the base data and can be used by database developers to create a physical database.
- It is also helpful to identify missing and redundant data.
- Though the initial creation of data model is labor and time consuming, in the long run, it makes your IT infrastructure upgrade and maintenance cheaper and faster.

Types of Data Models

There are mainly three different types of data models:

1. **Conceptual:** This Data Model defines **WHAT** the system contains. This model is typically created by Business stakeholders and Data Architects. The purpose is to organize, scope and define business concepts and rules.
2. **Logical:** Defines **HOW** the system should be implemented regardless of the DBMS. This model is typically created by Data Architects and Business Analysts. The purpose is to developed technical map of rules and data structures.
3. **Physical**: This Data Model describes **HOW** the system will be implemented using a specific DBMS system. This model is typically created by DBA and developers. The purpose is actual implementation of the database.



Conceptual Model

The main aim of this model is to establish the entities, their attributes, and their relationships. In this Data modeling level, there is hardly any detail available of the actual Database structure.

The 3 basic tenants of Data Model are

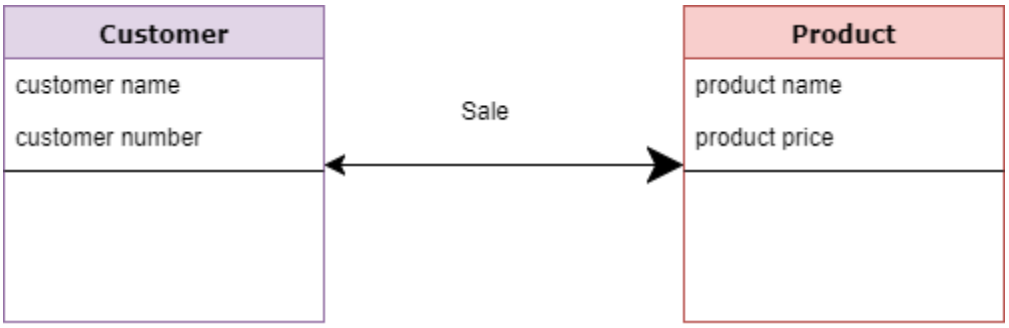
Entity: A real-world thing

Attribute: Characteristics or properties of an entity

Relationship: Dependency or association between two entities

For example:

- Customer and Product are two entities. Customer number and name are attributes of the Customer entity
- Product name and price are attributes of product entity
- Sale is the relationship between the customer and product



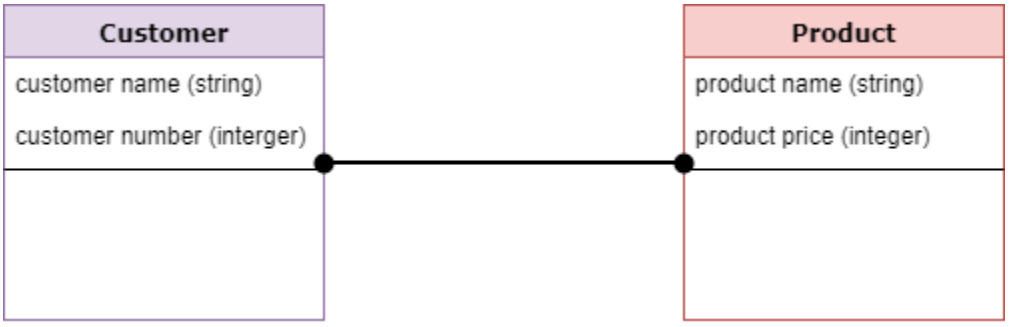
Characteristics of a conceptual data model

- Offers Organisation-wide coverage of the business concepts.
- This type of Data Models are designed and developed for a business audience.
- The conceptual model is developed independently of hardware specifications like data storage capacity, location or software specifications like DBMS vendor and technology. The focus is to represent data as a user will see it in the "real world."

Conceptual data models known as Domain models create a common vocabulary for all stakeholders by establishing basic concepts and scope.

Logical Data Model

Logical data models add further information to the conceptual model elements. It defines the structure of the data elements and set the relationships between them.



The advantage of the Logical data model is to provide a foundation to form the base for the Physical model. However, the modeling structure remains generic.

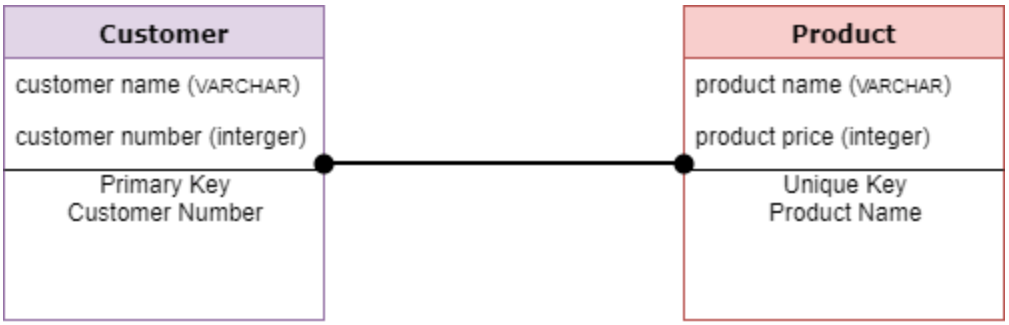
At this Data Modeling level, no primary or secondary key is defined. At this Data modeling level, you need to verify and adjust the connector details that were set earlier for relationships.

Characteristics of a Logical data model

- Describes data needs for a single project but could integrate with other logical data models based on the scope of the project.
- Designed and developed independently from the DBMS.
- Data attributes will have datatypes with exact precisions and length.
- Normalization processes to the model is applied typically till 3NF.

Physical Data Model

A Physical Data Model describes the database specific implementation of the data model. It offers an abstraction of the database and helps generate schema. This is because of the richness of meta-data offered by a Physical Data Model.



This type of Data model also helps to visualize database structure. It helps to model database columns keys, constraints, indexes, triggers, and other RDBMS features.

Characteristics of a physical data model:

- The physical data model describes data need for a single project or application though it maybe integrated with other physical data models based on project scope.
- Data Model contains relationships between tables that which addresses cardinality and nullability of the relationships.
- Developed for a specific version of a DBMS, location, data storage or technology to be used in the project.
- Columns should have exact datatypes, lengths assigned and default values.
- Primary and Foreign keys, views, indexes, access profiles, and authorizations, etc. are defined.

Advantages and Disadvantages of Data Model:

Advantages of Data model:

- The main goal of a designing data model is to make certain that data objects offered by the functional team are represented accurately.
- The data model should be detailed enough to be used for building the physical database.

- The information in the data model can be used for defining the relationship between tables, primary and foreign keys, and stored procedures.
- Data Model helps business to communicate the within and across organizations.
- Data model helps to documents data mappings in ETL process
- Help to recognize correct sources of data to populate the model

Disadvantages of Data model:

- To develop Data model one should know physical data stored characteristics.
- This is a navigational system produces complex application development, management. Thus, it requires a knowledge of the biographical truth.
- Even smaller change made in structure require modification in the entire application.
- There is no set data manipulation language in DBMS.

Conclusion

- Data modeling is the process of developing data model for the data to be stored in a Database.
- Data Models ensure consistency in naming conventions, default values, semantics, security while ensuring quality of the data.
- Data Model structure helps to define the relational tables, primary and foreign keys and stored procedures.
- There are three types of conceptual, logical, and physical.
- The main aim of conceptual model is to establish the entities, their attributes, and their relationships.
- Logical data model defines the structure of the data elements and set the relationships between them.
- A Physical Data Model describes the database specific implementation of the data model.
- The main goal of a designing data model is to make certain that data objects offered by the functional team are represented accurately.
- The biggest drawback is that even smaller change made in structure require modification in the entire application.

Flow-Oriented Modeling

The requirements model has many different dimensions. Flow oriented models, behavioral models, and the special requirements analysis considerations that come into play when WebApps are developed. Each of these modeling representations supplements the use cases, data models, and class based models.

Although the data flow diagram (DFD) and related diagrams and information are not a formal part of UML, they can be used to complement UML diagrams and provide additional insight into system requirements andflow

The DFD takes an input-process-output view of a system. That is, data objects flow into the software, are transformed by processing elements, and resultant data objects flow out of the software. Data objects are represented by labeled arrows, and transformations are represented by circles (also called bubbles). The DFD is presented in a hierarchical fashion. That is, the first data flow model (sometimes called a level 0 DFD or context diagram) represents the system as a whole. Subsequent data flow diagrams refine the context diagram, providing increasing detail with each subsequentlevel.

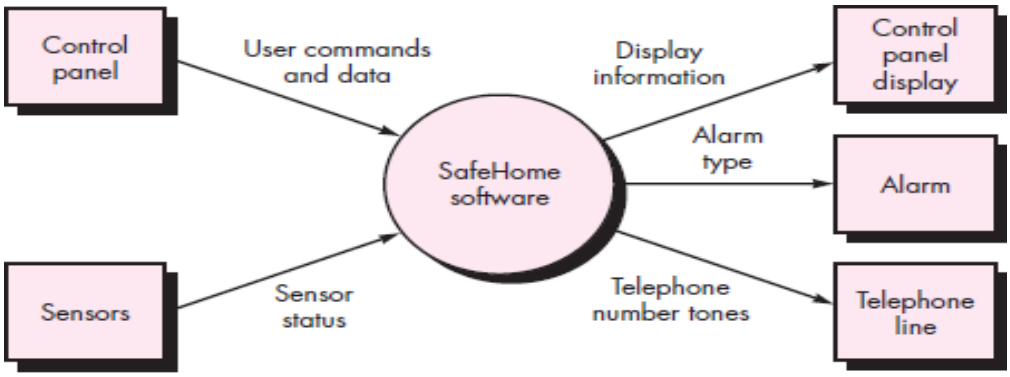


Fig 7.1: Context level DFD for safeHome security function

Creating a Data Flow Model: The data flow diagram enables you to develop models of the information domain and functional domain. As the DFD is refined into greater levels of detail, you perform an implicit functional decomposition of thesystem.

A few simple guidelines of a data flow diagram:

(1) the level 0 data flow diagram should depict the software/system as a singlebubble;

(2) primary input and output should be carefullynoted;

(3) refinement should begin by isolating candidate processes, data objects, and data stores to be represented at the nextlevel;

(4) all arrows and bubbles should be labeled with meaningfulnames;

(5) information flow continuity must be maintained from level to level,2and

(6) one bubble at a time should be refined. There is a natural tendency to overcomplicate the data flow diagram. This occurs when you attempt to show too much detail too early or represent procedural aspects of the software in lieu of information flow.

2.9.1.1 Creating a Control Flow Model: For some types of applications, the data model and the data flow diagram are all that is necessary to obtain meaningful insight into software requirements. However, a large class of applications are "driven" by events rather than data, produce control information rather than reports or displays, and process information with heavy concern for time and performance. Such applications require the use of control flow modeling in addition to data flowmodeling.

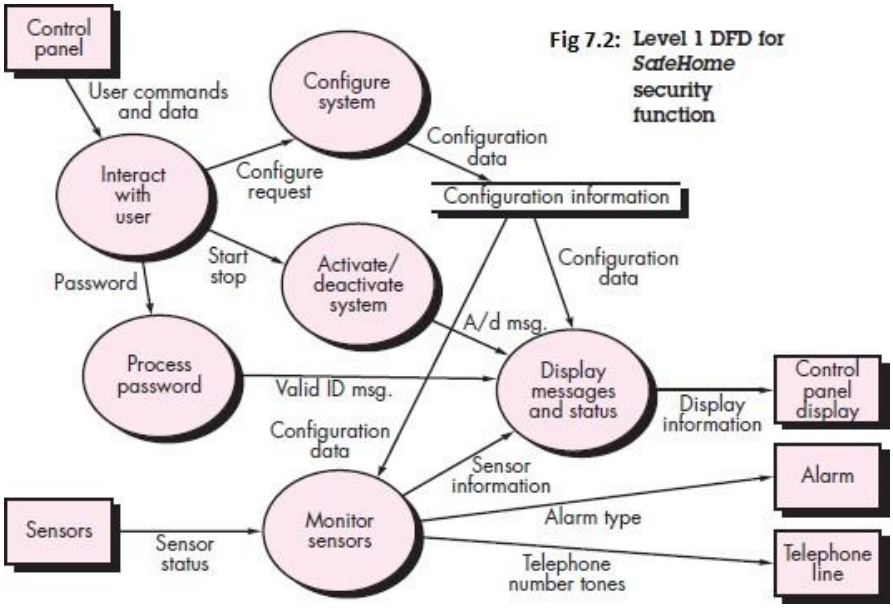


Fig 7.2: Level 1 DFD for *SafeHome* security function

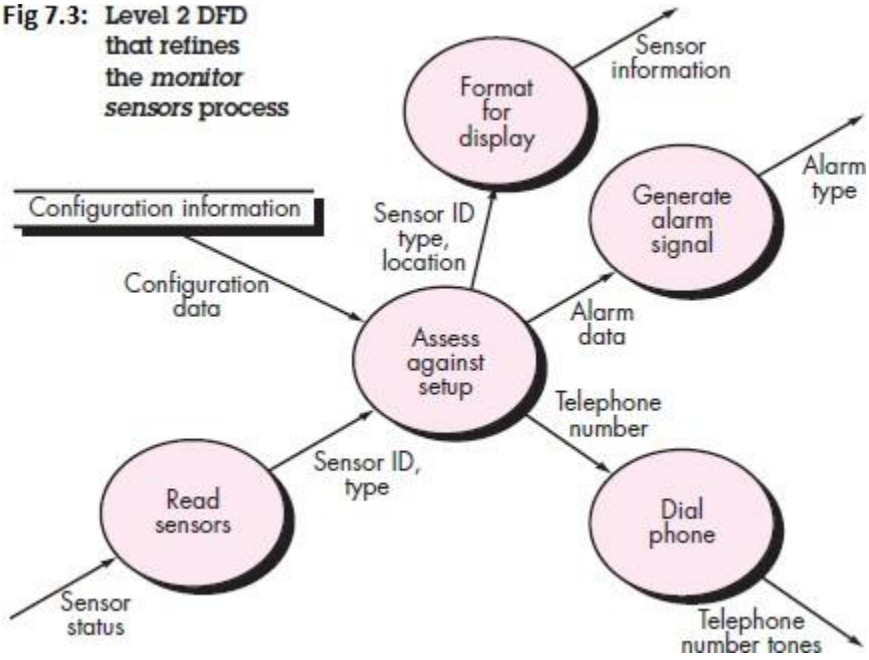


Fig 7.3: Level 2 DFD that refines the *monitor sensors* process

An event or control item is implemented as a Boolean value (e.g., true or false, on or off, 1 or 0) or a discrete list of conditions (e.g., empty, jammed, full). To select potential candidate events, the following guidelines are suggested:

- List all sensors that are "read" by thesoftware.
- List all interruptconditions.
- List all "switches" that are actuated by anoperator.
- List all dataconditions.

• Recalling the noun/verb parse that was applied to the processing narrative, review all "control items" as possible control specificationinputs/outputs.
• Describe the behavior of a system by identifying its states, identify how each state is reached, and define the transitions betweenstates.
• Focus on possible omissions—a very common error in specifyingcontrol

2.9.1.2 The Control Specification: A control specification (CSPEC) represents the behavior of the system in two different ways. The CSPEC contains a state diagram that is a sequential specification of behavior. It can also contain a program activation table—a combinatorial specification of behavior.
Figure 7.4 depicts a preliminary state diagram for the level 1 control flow model for SafeHome.

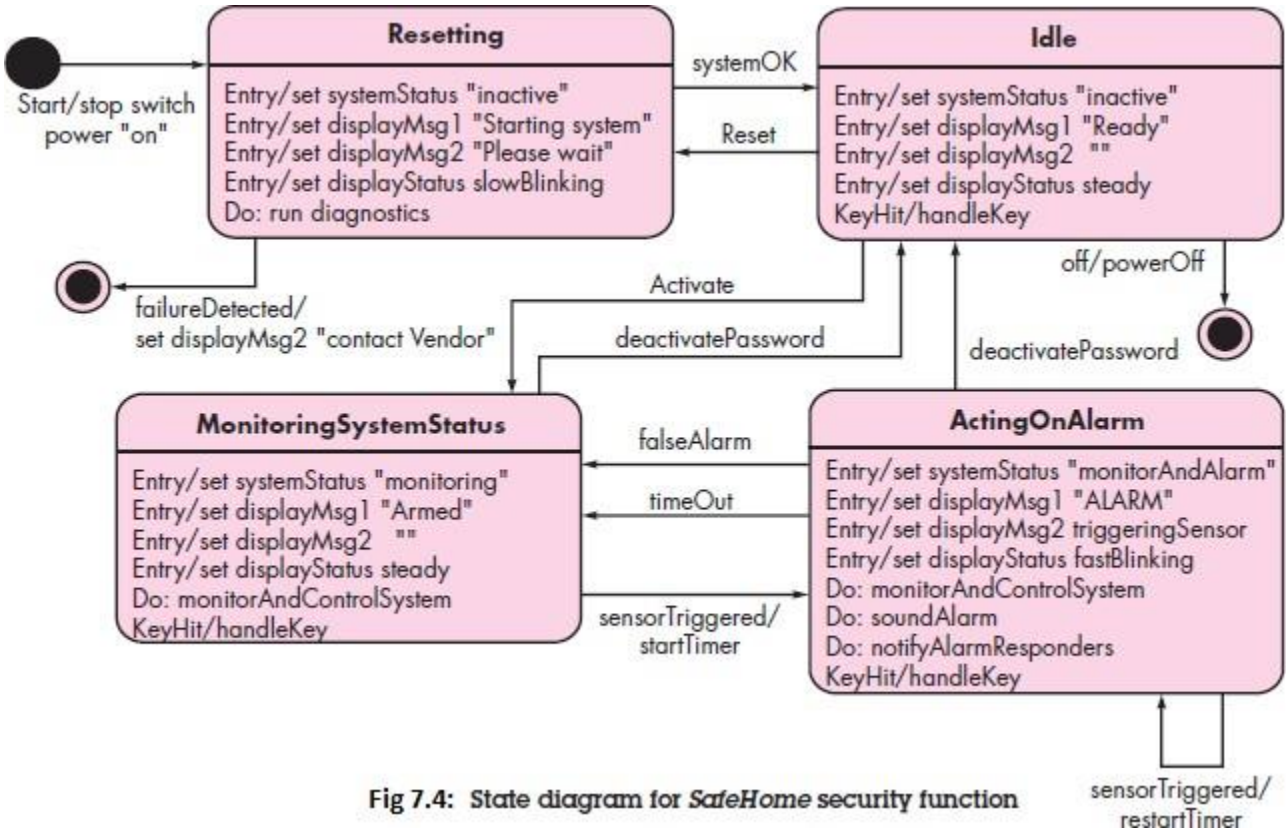


Fig 7.4: State diagram for *SafeHome* security function

2.9.1.3 The Process Specification: *The PSPEC is a "minispecification" for each transform at the lowest refined level of a DFD.*The process specification (PSPEC) is used to describe all flow model processes that appear at the final level of refinement. The content of the process specification can include narrative text, a program design language (PDL) description of the process algorithm, mathematical equations, tables, or UML activitydiagrams.

input events						
sensor event	0	0	0	0	1	0
blink flag	0	0	1	1	0	0
start stop switch	0	1	0	0	0	0
display action status complete	0	0	0	1	0	0
in-progress	0	0	1	0	0	0
time out	0	0	0	0	0	1
output						
alarm signal	0	0	0	0	1	0
process activation						
monitor and control system	0	1	0	0	1	1
activate/deactivate system	0	1	0	0	0	0
display messages and status	1	0	1	1	1	1
interact with user	1	0	0	1	0	1

Fig 7.5: Process activation table for SafeHome security function

2.9.2 CREATING A BEHAVIORALMODEL

We can represent the behavior of the system as a function of specific events and time.
The behavioral model indicates how software will respond to external events or stimuli. To create the model, you should perform the followingsteps:

1. Evaluate all use cases to fully understand the sequence of interaction within thesystem.
2. Identify events that drive the interaction sequence and understand how these events relate to specificobjects.
3. Create a sequence for each usecase.
4. Build a state diagram for thesystem.
5. Review the behavioral model to verify accuracy andconsistency.

2.9.2.1 Identifying Events with the Use Case: The use case represents a sequence of activities that involves actors and the system. In general, an event occurs whenever the system and an actor exchange information. A use case is examined for points of informationexchange.

The underlined portions of the use case scenario indicate events. An actor should be identified for each event; the information that is exchanged should be noted, and any conditions or constraints should be listed. Once all events have been identified, they are allocated to the objects involved. Objects can be responsible for generating events or recognizing events that have occurred elsewhere.

2.9.2.2 State Representations: In the context of behavioral modeling, two different characterizations of states must beconsidered:
(1) The state of each class as the system performs its functionand
(2) The state of the system as observed from the outside as the system performs itsfunction.

The state of a class takes on both passive and active characteristics. A passive state is simply the current status of all of an object's attributes. The active state of an object indicates the current status of the object as it undergoes a continuing transformation or processing. An event (sometimes called a trigger) must occur to force an object to make a transition from one active state to another. Two different behavioral representations are discussed in the paragraphs that follow.

1) The first indicates how an individual class changes state based on externalevents
2) The second shows the behavior of the software as a function oftime.

State diagrams for analysis classes. One component of a behavioral model is

a UML state diagram9 that represents active states for each class and the events (triggers) that cause changes between these active states. Figure 7.6 illustrates a state diagram
for the ControlPanel object in the SafeHome security function.
Each arrow shown in Figure 7.6 represents a transition from one active state of an object to another. The labels shown for each arrow represent the event that triggers the transition.

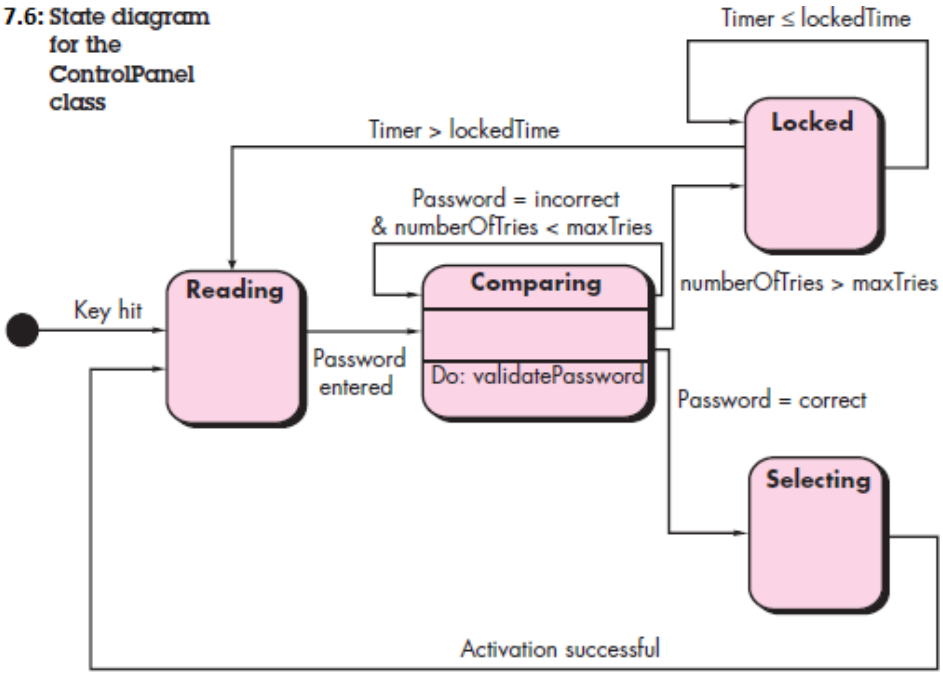
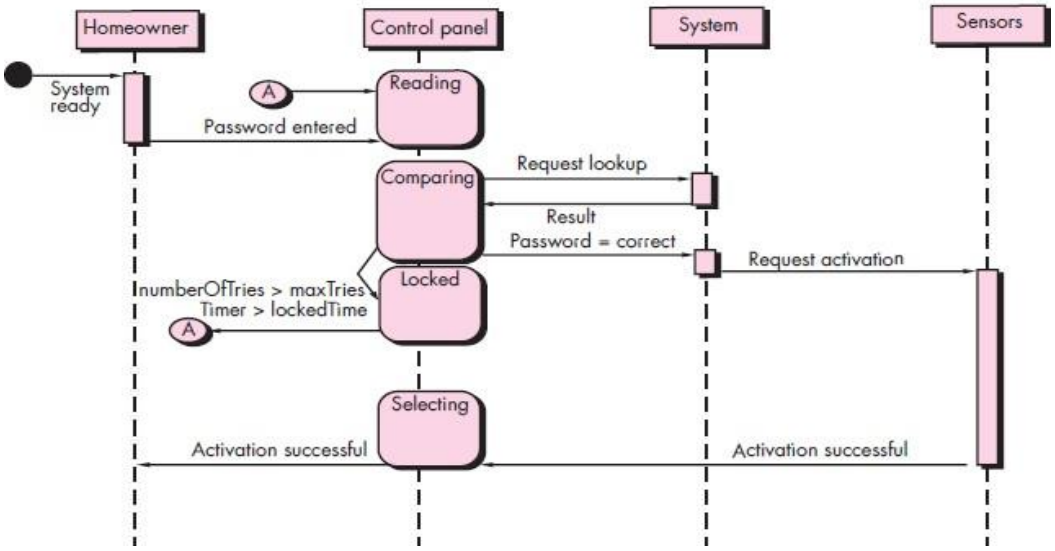


Fig 7.6: State diagram for the ControlPanel class

Sequence diagrams. The second type of behavioral representation, called a sequence diagram in UML, indicates how events cause transitions from object to object. Once events have been identified by examining a use case, the modeler creates a sequence diagram—a representation of how events cause flow from one object to another as a function of time. In essence, the sequence diagram is a shorthand version of the use case. It represents key classes and the events that cause behavior to flow from class to class. Figure 7.7 illustrates a partial sequence diagram for the SafeHome security function.

Fig 7.7: Sequence diagram (partial) for the *SafeHome* security function



Class-based Modeling,

Class-based modeling represents the objects that the system will manipulate, the operations that will be applied to the objects to effect the manipulation, relationships between the objects, and the collaborations that occur between the classes that are defined. The elements of a class- based model include classes and objects, attributes, operations, class responsibility, collaborator (CRC) models, collaboration diagrams, and packages.

2.0.1.1 Identifying AnalysisClasses

Classes are determined by underlining each *noun*or *nounphrase*and entering it into a simple table. If the class (noun) is required to implement a solution, then it is part of the solution space; otherwise, if a class is necessary only to describe a solution, it is part of the problem space.

Analysis classes manifest themselves in one of the following ways:

- *Externalentities* that produce or consume information to be used by a computer-based system.
- *Things*(e.g., reports, displays, letters, signals) that are part of the information domain for the problem.
- *Occurrences or events*(e.g., a property transfer or the completion of a series of robot movements) that occur within the context of systemoperation.
- *Roles*(e.g., manager, engineer, salesperson) played by people who interact with thesystem.
- *Organizational units*(e.g., division, group, team) that are relevant to anapplication.
- *Places*(e.g., manufacturing floor or loading dock) that establish the context of the problem and the overall function of thesystem.
- *Structures*(e.g., sensors, four-wheeled vehicles, or computers) that define a class of objects or related

classes ofobjects.

2.0.1.2 Specifying Attributes: Attributes are the set of data objects that fully define the class within the context of the problem. Attributes describe a class that has been selected for inclusion in the requirements model. In essence, it is the attributes that define the class—that clarify what is meant by the class in the context of the problemspace.
To develop a meaningful set of attributes for an analysis class, you should study each use case and select those "things" that reasonably "belong" to the class.

2.0.1.3 Defining Operations: Operations define the behavior of an object. Although many different types of operations exist, they can generally be divided into four broad categories: (1) operations that manipulate data in some way. (2) operations that perform a computation, (3) operations that inquire about the state of an object, and (4) operations that monitor an objectfor

the occurrence of a controlling event. These functions are accomplished by operating on attributes and/or associations. Therefore, an operation must have "knowledge" of the nature of the class' attributes and associations.
As a first iteration at deriving a set of operations for an analysis class, you can again study a processing narrative (or use case) and select those operations that reasonably belong to the class. To accomplish this, the grammatical parse is again studied and verbs are isolated. Some of these verbs will be legitimate operations and can be easily connected to a specific class.

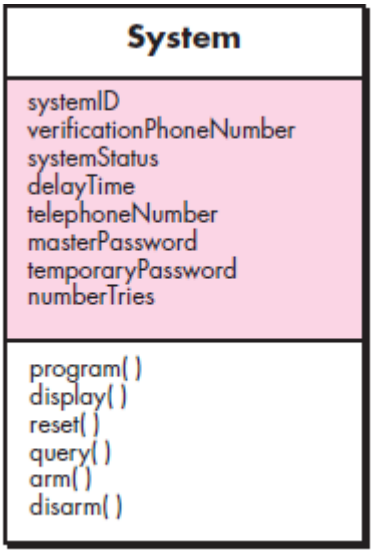


Fig 6.9: Class diagram for the System class

2.0.1.4 Class-Responsibility-Collaborator (CRC) Modeling: Class-responsibility-collaborator (CRC) modeling provides a simple means for *identifying*and *organizing*the classes that are relevant to system or productrequirements.

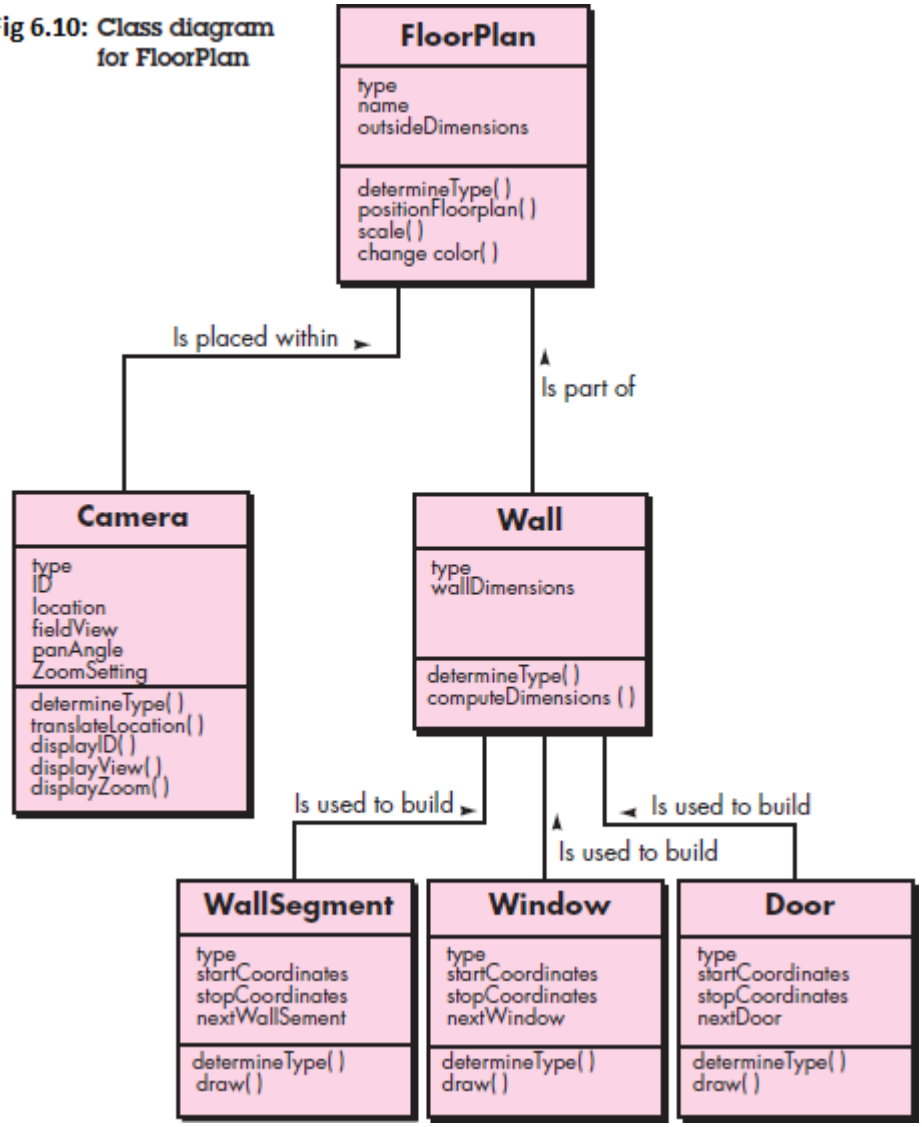
One purpose of CRC cards is to fail early, to fail often, and to fail inexpensively. It is a lot cheaper to tear up a bunch of cards than it would be to reorganize a large amount of source code.

Ambler describes CRC modeling in the following way:
A CRC model is really a collection of standard index cards that represent classes. The cards are divided into three sections. Along the top of the card you write the *nameoftheclass*. In the body of the card you list the *classresponsibilities*on the left and the *collaborators*on the right.

In reality, the CRC model may make use of actual or virtual index cards. The intent is to develop an organized representation of classes. Responsibilities are the attributes and operations that are relevant for the class. Stated simply, a responsibility is "anything the class knows or does". Collaborators are those classes that are required to provide a class with the information needed to complete a responsibility.
In general, a collaboration implies either a request for information or a request for some action.

A simple CRC index card for the FloorPlan class is illustrated in Figure 6.11. The list of responsibilities shown on the CRC card is preliminary and subject to additions or modification. The classes Wall and Camera are noted next to the responsibility that will require their collaboration.

Fig 6.10: Class diagram for FloorPlan

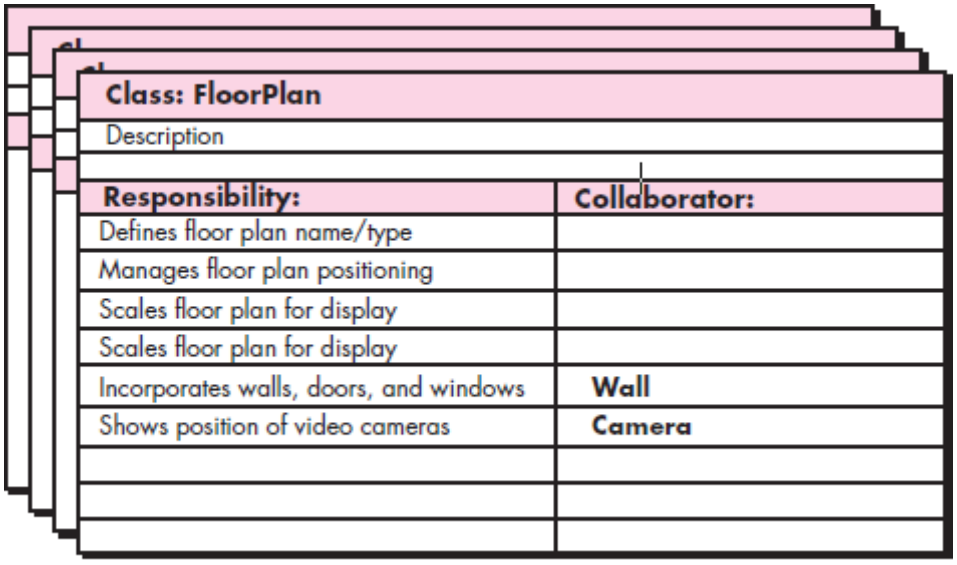


Classes. Basic guidelines for identifying classes and objects were presented earlier in this chapter. The taxonomy of class types presented in Section 6.5.1 can be extended by considering the followingcategories:

• *Entity classes*, also called model or business classes, are extracted directly from the statement of the problem These classes typically represent things that are to be stored in a database and persist throughout the duration of theapplication.

• *Boundary classes*are used to create the interface (e.g., interactive screen or printedreports) that the user sees and interacts with as the software is used. Entity objects containinformation

that is important to users, but they do not display themselves. Boundary classes are designed with the responsibility of managing the way entity objects are represented to users.



Class: FloorPlan	
Description	
Responsibility:	**Collaborator:**
Defines floor plan name/type	
Manages floor plan positioning	
Scales floor plan for display	
Scales floor plan for display	
Incorporates walls, doors, and windows	**Wall**
Shows position of video cameras	**Camera**

Fig 6.11: A CRC model index card

• *Controller classes*manage a "unit of work" from start to finish. That is, controller classes can be designed to manage (1) the creation or update of entity objects, (2) the instantiation of boundary objects as they obtain information from entity objects, (3) complex communication between sets of objects, (4) validation of data communicated between objects or between the user and the application. In general, controller classes are not considered until the design activity hasbegun.

Responsibilities. The five guidelines for allocating responsibilities to classes:

1. System intelligence should be distributed across classes to best address the needs of the problem. Every application encompasses a certain degree of intelligence; that is, what the system knows and what it can do. This intelligence can be distributed acrosss classes in a umber of different ways. "Dumb" classes can be modeled to act as servants to a few "smart" classes.

To determine whether system intelligence is properly distributed, the responsibilities noted on each CRC model index card should be evaluated to determine if any class has an extraordinarily long list of responsibilities. Example is aggregation of classes.

2. Each responsibility should be stated as generally as possible. This guideline implies that general responsibilities (both attributes and operations) should reside high in the classhierarchy

3. Information and the behavior related to it should reside within the same class. This achieves the object-oriented principle called encapsulation. Data and the processes that manipulate the data should be packaged as a cohesiveunit.

4. Information about one thing should be localized with a single class, not distributed across multiple classes. A single class should take on the responsibility for storing and manipulatingaspecifictypeofinformation.Thisresponsibilityshouldnot,ingeneral,beshared

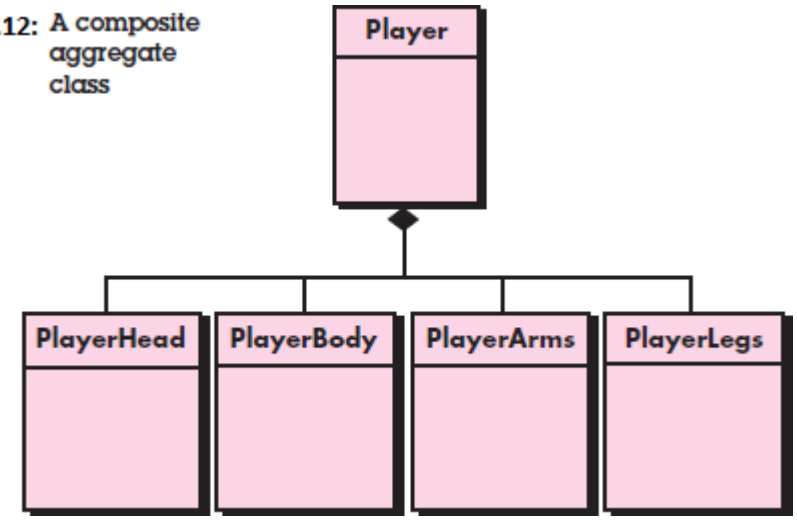
across a number of classes. If information is distributed, software becomes more difficult to maintain and more challenging to test.

5. Responsibilities should be shared among related classes, when appropriate. There are many cases in which a variety of related objects must all exhibit the same behavior at the same time.

Collaborations. Classes fulfill their responsibilities in one of two ways: (1) A class can use its own operations to manipulate its own attributes, thereby fulfilling a particular responsibility, or (2) a class can collaborate with other classes.

To help in the identification of collaborators, you can examine three different generic relationships between classes (1) the is-part-of relationship, (2) the has-knowledge-of relationship, and (3) the depends-uponrelationship.

Fig 6.12: A composite aggregate class



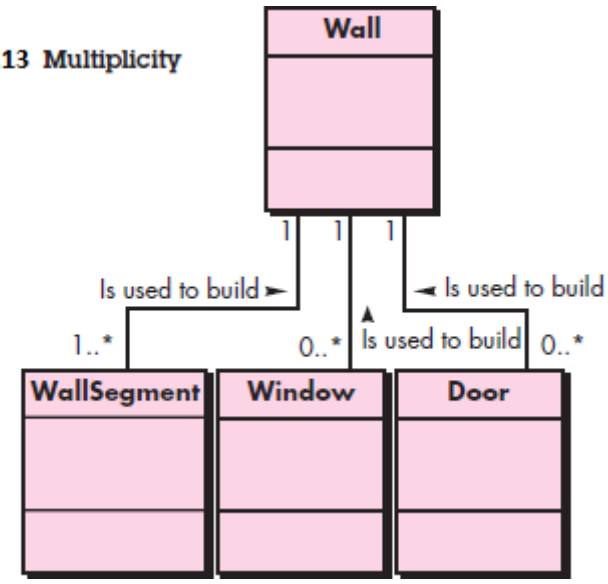
When a complete CRC model has been developed, stakeholders can review the model using the followingapproach

1. All participants in the review are given a subset of the CRC model index cards. Cards that collaborate should beseparated
2. All use-case scenarios (corresponding use-case diagrams) should be organized into categories.
3. The review leader reads the use casedeliberately.
4. When the token is passed, the holder of the Sensor card is asked to describe the responsibilities noted on thecard.
5. If the responsibilities and collaborations noted on the index cards cannot accommodate the use case, modifications are made to thecards.

This continues until the use case is finished. When all use cases have been reviewed, requirements modeling continue.

2.0.1.5 Associations and Dependencies: In many instances, two analysis classes are related to one another in some fashion, much like two data objects may be related to one another. In UML these relationships are called*associations*.
In some cases, an association may be further defined by indicating multiplicity. Referring to The multiplicity constraints are illustrated in Figure 6.13, where "one or more" is represented using 1. .*, and "0 or more" by 0 . .*. In UML, the asterisk indicates an unlimited upper bound on the range.

Fig 6.13 Multiplicity



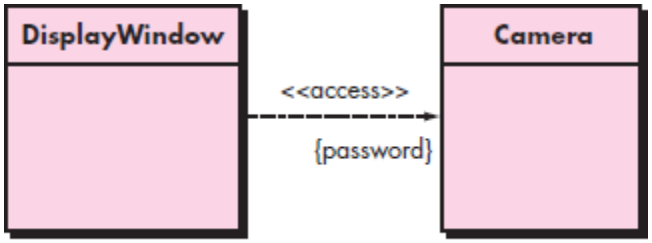


Fig 6.14: Dependencies

2.0.1.6 Analysis Packages: An important part of analysis modeling is *categorization*. Thatis, various elements of the analysis model are categorized in a manner that *packages*them as a grouping—called an *analysis package*—that is given a representativename.

For example, Classes such as Tree, Landscape, Road, Wall, Bridge, Building, and VisualEffect might fall within this category. Others focus on the characters within the game, describing their physical features, actions, and constraints.
These classes can be grouped in analysis packages as shown in Figure 6.15. The *plussign*preceding the analysis class name in each package indicates that the classes have public visibility and are therefore accessible from other packages.

Although they are not shown in the figure, other symbols can precede an element within a package. A *minussign*indicates that an element is hidden from all other packages and a *#symbol*indicates that an element is accessible only to packages contained within a given package.

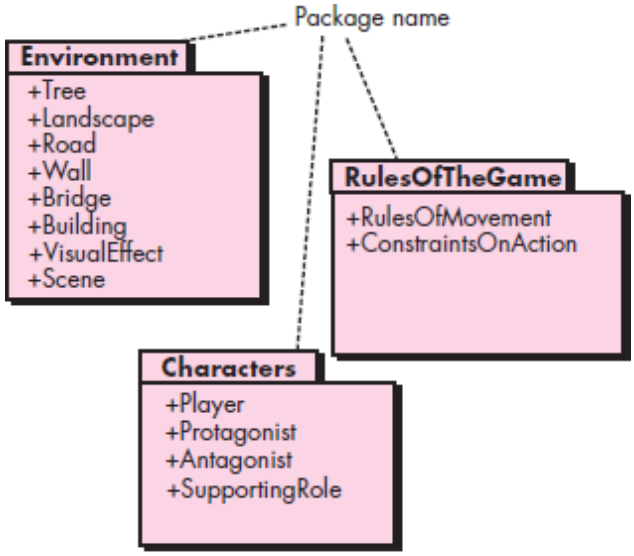


Fig 6.15: Packages

Metrics for Analysis Models

Software patterns are a mechanism for capturing domain knowledge in a way that allows it to be reapplied when a new problem is encountered. In some cases, the domain knowledge is applied to a new problem within the same application domain.

Once the pattern has been discovered, it is documented by describing "explicitly the general problem to which the pattern is applicable, the prescribed solution, assumptions and constraints of using the pattern in practice, and often some other information about the pattern, such as the *motivation and driving forces for using the pattern, discussion of the pattern's advantages and disadvantages, and references to some known examples of using that pattern in practical applications*".

The pattern can be reused when performing requirements modeling for an application within a domain. Once an appropriate pattern is selected, it is integrated into the requirements model by reference to the pattern name.

2.9.1 Discovering Analysis Patterns: The requirements model is comprised of a wide variety of elements: scenario-based (use cases), data-oriented (the data model), class-based, flow- oriented, andbehavioral.

Each of these elements examines the problem from a different perspective, and each provides an opportunity to discover patterns that may occur throughout an application domain, or by analogy, across different applicationdomains.

The most basic element in the description of a requirements model is the use case. A semantic analysis pattern (SAP) "is a pattern that describes a small set of coherent use cases that together describe a basic genericapplication".

Consider the following preliminary use case for software required to control and monitor a real-view camera and proximity sensor for an automobile:
Use case: Monitor reverse motion

Description: When the vehicle is placed in reverse gear, the control software enables a video feed from a rear-placed video camera to the dashboard display.

This use case implies a variety of functionality that would be refined and elaborated during requirements gathering and modeling.

2.9.2 A Requirements Pattern Example: Actuator-Sensor One of the requirements of the SafeHome security function is the ability to monitory securitysensors.
Pattern Name. Actuator-Sensor

Intent. Specify various kinds of sensors and actuators in an embedded system.
Motivation. Embedded systems usually have various kinds of sensors and actuators. These sensors and actuators are all either directly or indirectly connected to a control unit.
Constraints
• Each passive sensor must have some method to read sensor input and attributes that represent the sensorvalue.
• Each active sensor must have capabilities to broadcast update messages when its value changes.
• Each active sensor should send a life tick, a status message issued within a specified time frame, to detectmalfunctions.
Applicability. Useful in any system in which multiple sensors and actuators are present.
Structure. A UML class diagram for the Actuator-Sensor pattern is shown in Figure 7.8.

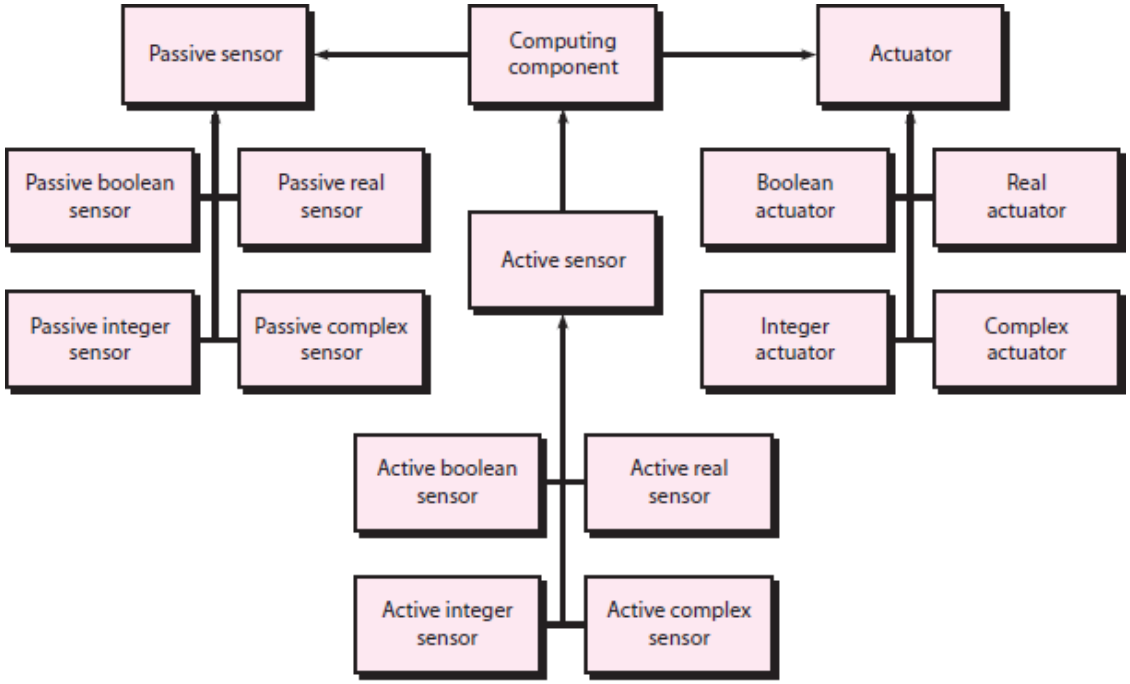


Fig 7.8: UML sequence diagram for the Actuator-Sensor pattern.

Behavior. Figure 7.9 presents a UML sequence diagram for an example of the Actuator-Sensor pattern as it might be applied for the SafeHome function that controls the positioning (e.g., pan, zoom) of a security camera.

Participants. Patterns description "itemizes the classes/objects that are included in the requirements pattern" and describes the responsibilities of each class/object (Figure 7.8). An abbreviated list follows:

• PassiveSensor abstract: Defines an interface for passivesensors.

• PassiveBooleanSensor: Defines passive Booleansensors.

• PassiveIntegerSensor: Defines passive integersensors.

Collaborations. This describes how objects and classes interact with one another and how each carries out itsresponsibilities.

• When the ComputingComponent needs to update the value of a PassiveSensor, it queries the sensors, requesting the value by sending the appropriatemessage.

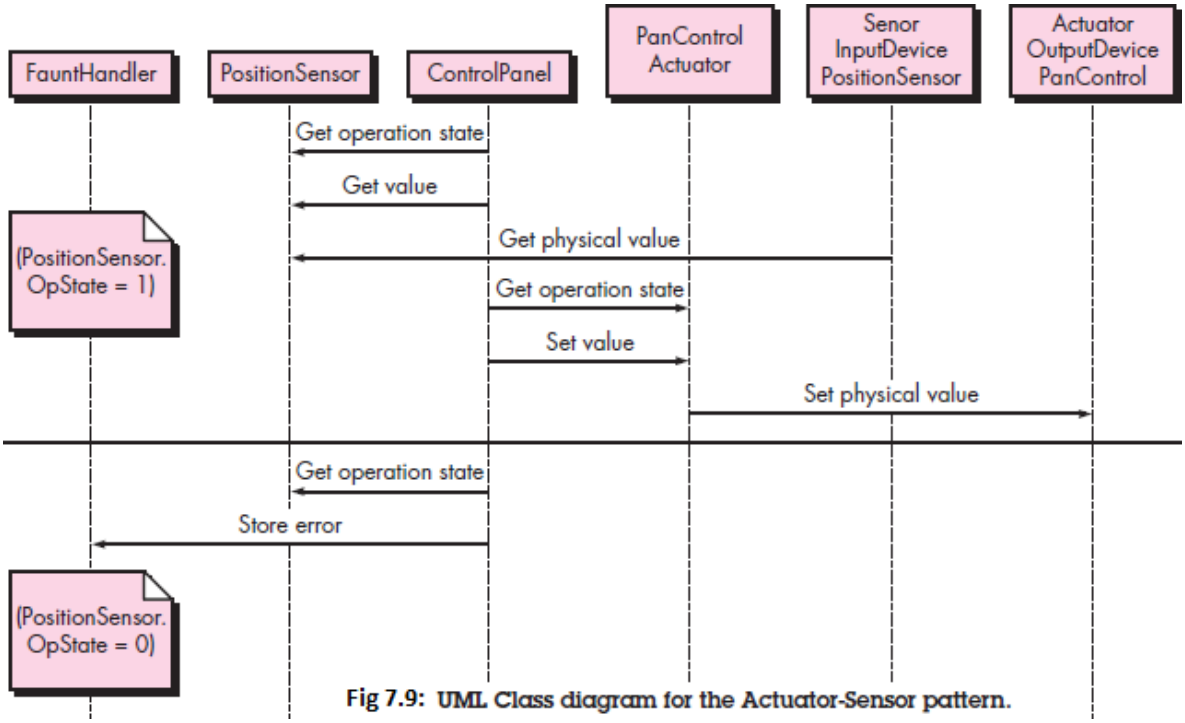


Fig 7.9: UML Class diagram for the Actuator-Sensor pattern.

Consequences

1. Sensor and actuator classes have a commoninterface.
2. Class attributes can only be accessed through messages, and the class decides whether or not to accept the message. For example, if a value of an actuator is set above a maximum value, then the actuator class may not accept the message, or it might use a default maximum value.
3. The complexity of the system is potentially reduced because of the uniformity of interfaces for actuators andsensors.

2.10 REQUIREMENTS MODELING FORWEBAPPS

The Web development process must be agile, and analysis is time consuming

2.10.1 How Much Analysis Is Enough?: The degree to which requirements modeling for WebApps are emphasized depends on the followingfactors:

- Size and complexity of WebAppincrement.
- Number ofstakeholders
- Size of the WebAppteam.
- Degree to which members of the WebApp team have worked togetherbefore
- Degree to which the organization's success is directly dependent on the success of the WebApp.

2.10.2 Requirements Modeling Input: An agile version of the generic software process can be applied when WebApps are engineered. The process incorporates a communication activitythat

identifies stakeholders and user categories, the business context, defined informational and applicative goals, general WebApp requirements, and usage scenarios—information that becomes input to requirements modeling. This information is represented in the form of natural language descriptions, rough outlines, sketches, and other informal representations.

To summarize, the inputs to the requirements model will be the information collected during the communication activity—anything from an informal e-mail to a detailed project brief complete with comprehensive usage scenarios and productspecifications.

2.10.3 Requirements Modeling Output: Requirements analysis provides a disciplined mechanism for representing and evaluating WebApp content and function. While the specific models depend largely upon the nature of the WebApp, there are five main classes ofmodels:

• Content model—identifies the full spectrum of content to be provided by the WebApp. Content includes text, graphics and images, video, and audiodata.

• Interaction model—describes the manner in which users interact with theWebApp.

• Functional model—defines the operations that will be applied to WebApp content and describes other processing functions that are independent of content but necessary to the end user.

• Navigation model—defines the overall navigation strategy for theWebApp.

• Configuration model—describes the environment and infrastructure in which the WebApp resides.

2.10.4 Content Model for WebApps: The content model contains structural elements that provide an important view of content requirements for a WebApp. These structural elements encompass content objects and all analysis classes—user-visible entities that are created or manipulated as a user interacts with theWebApp.

Content can be developed prior to the implementation of the WebApp, while the WebApp is being built, or long after the WebApp is operational. In every case, it is incorporated via navigational reference into the overall WebApp structure. A content object might be a textual description of a product, an article describing a news event, an action photograph taken at a sporting event, a user's response on a discussion forum, an animated representation of a corporate logo, a short video of a speech, or an audio overlay for a collection of presentation slides. The content objects might be stored as separate files, embedded directly into Web pages, or obtained dynamically from a database. In other words, a content object is any item of cohesive information that is to be presented to an end user. Content objects can be determined directly from use cases by examining the scenario description for direct and indirect references to content. Refer to figure 7.10 for contentmodeling.

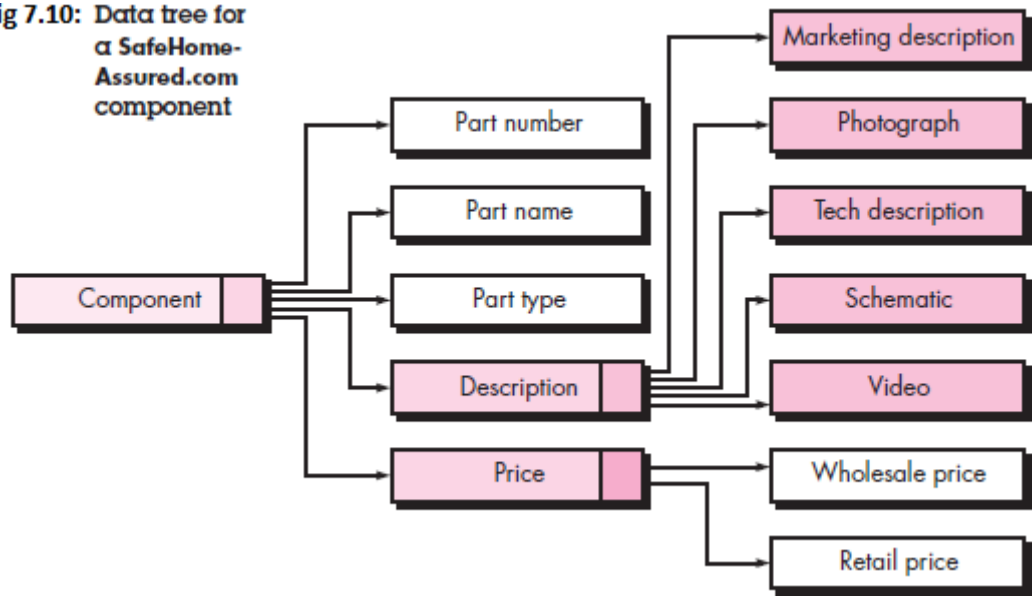


Fig 7.10: Data tree for a SafeHome-Assured.com component

A data tree can be created for any content that is composed of multiple content objects and data items. The data tree is developed in an effort to define hierarchical relationships among content objects and to provide a means for reviewing content so that omissions and inconsistencies are uncovered before design commences. In addition, the data tree serves as the basis for content design.

2.10.5 Interaction Model for WebApps: The vast majority of WebApps enable a "conversation" between an end user and application functionality, content, and behavior. This conversation can be described using an interaction model that can be composed of one or more of the following elements: (1) use cases, (2) sequence diagrams, (3) state diagrams,16 and/or (4) user interface prototypes.
In many instances, a set of use cases is sufficient to describe the interaction at an analysis level. However, when the sequence of interaction is complex and involves multiple analysis classes or many tasks, it is sometimes worthwhile to depict it using a more rigorous diagrammatic form. Because WebApp construction tools are plentiful, relatively inexpensive, and functionally powerful, it is best to create the interface prototype using such tools.

2.10.6 Functional Model for WebApps: Many WebApps deliver a broad array of computational and manipulative functions that can be associated directly with content and that are often a major goal of user-WebApp interaction. For this reason, functional requirements must be analyzed, and when necessary,modeled.

The functional model addresses two processing elements of the WebApp, each representing a different level of procedural abstraction:
(1) user-observable functionality that is delivered by the WebApp to end users,and

(2) the operations contained within analysis classes that implement behaviors associated with the class.

User-observable functionality encompasses any processing functions that are initiated directly by the user. These functions may actually be implemented using operations within analysis classes, but from the point of view of the end user, the function is the visibleoutcome.
At a lower level of procedural abstraction, the requirements model describes the processing to be performed by analysis class operations. These operations manipulate class attributes and are involved as classes collaborate with one another to accomplish some requiredbehavior.

Regardless of the level of procedural abstraction, the UML activity diagram can be used to represent processing details.

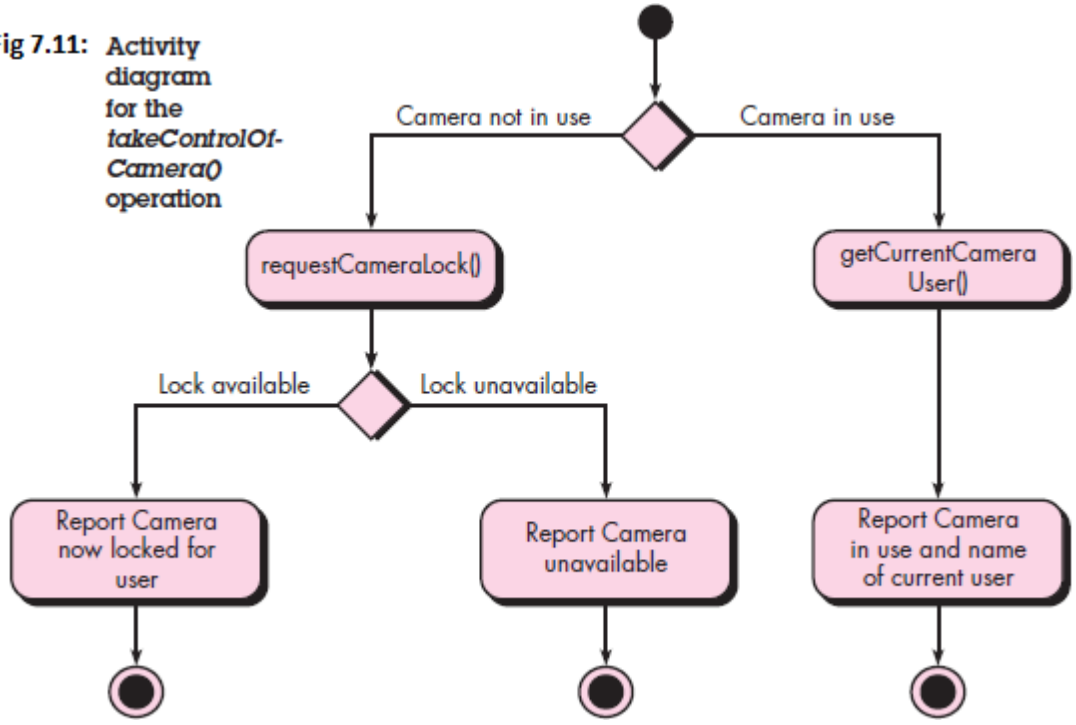


Fig 7.11: **Activity diagram for the *takeControlOf-Camera()* operation**

2.10.7 Configuration Models for WebApps: In some cases, the configuration model is nothing more than a list of server-side and client-side attributes. However, for more complex WebApps, a variety of configuration complexities (e.g., distributing load among multiple servers, caching architectures, remote databases, multiple servers serving various objects on the same Web page) may have an impact on analysis anddesign.
The UML deployment diagram can be used in situations in which complex configuration architectures must be considered.

2.10.8 Navigation Modeling: Navigation modeling considers how each user category will navigate from one WebApp element to another. The following questions should beconsidered:

• Should certain elements be easier to reach (require fewer navigation steps) than others? What is the priority forpresentation?
• Should certain elements be emphasized to force users to navigate in theirdirection?
• How should navigation errors behandled?
• Should navigation to related groups of elements be given priority over navigation to a specific element?
• Should navigation be accomplished via links, via search-based access, or by some other means?
• Should certain elements be presented to users based on the context of previous navigation actions?
• Should a navigation log be maintained forusers?
• Should a full navigation map or menu be available at every point in a user'sinteraction?
• Should navigation design be driven by the most commonly expected user behaviors or by the perceived importance of the defined WebAppelements?
• Can a user "store" his previous navigation through the WebApp to expedite futureusage?
• For which user category should optimal navigation bedesigned?
• How should links external to the WebApp be handled? Overlaying the existing browser window? As a new browser window? As a separateframe?