SOFTWARE ENGINEERING

CHAPTER-7
REQUIREMENTS MODELING:
FLOW, BEHAVIOR, PATTERNS, AND WEBAPPS

Software Engineering: A Practitioner's Approach, 7th edition
Originated by Roger S. Pressman

Software School, Fudan University
Spring Semester, 2016

Requirements Modeling Strategies

- One view of requirements modeling, called *structured analysis*, considers data and the processes that transform the data as separate entities
  - Data objects are modeled in a way that defines their attributes and relationships
  - Processes that manipulate data objects are modeled in a manner that shows how they transform data as data objects flow through the system
- A second approach to analysis modeled, called *object-oriented analysis*, focuses on
  - the definition of classes and
  - the manner in which they collaborate with one another to effect customer requirements

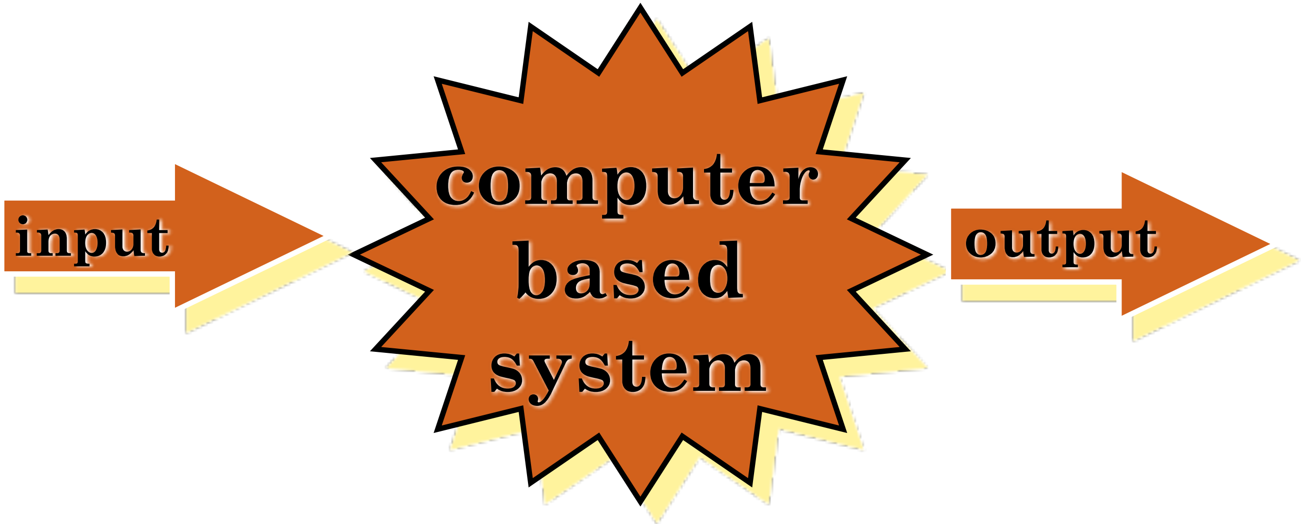
FLOW-ORIENTED MODELING

Flow-Oriented Modeling

- Represents how data objects are transformed at they move through the system
- **data flow diagram (DFD)** is the diagrammatic form that is used
- Considered by many to be an “old school” approach, but continues to provide a view of the system that is unique—it should be used to supplement other analysis model elements

THE FLOW MODEL

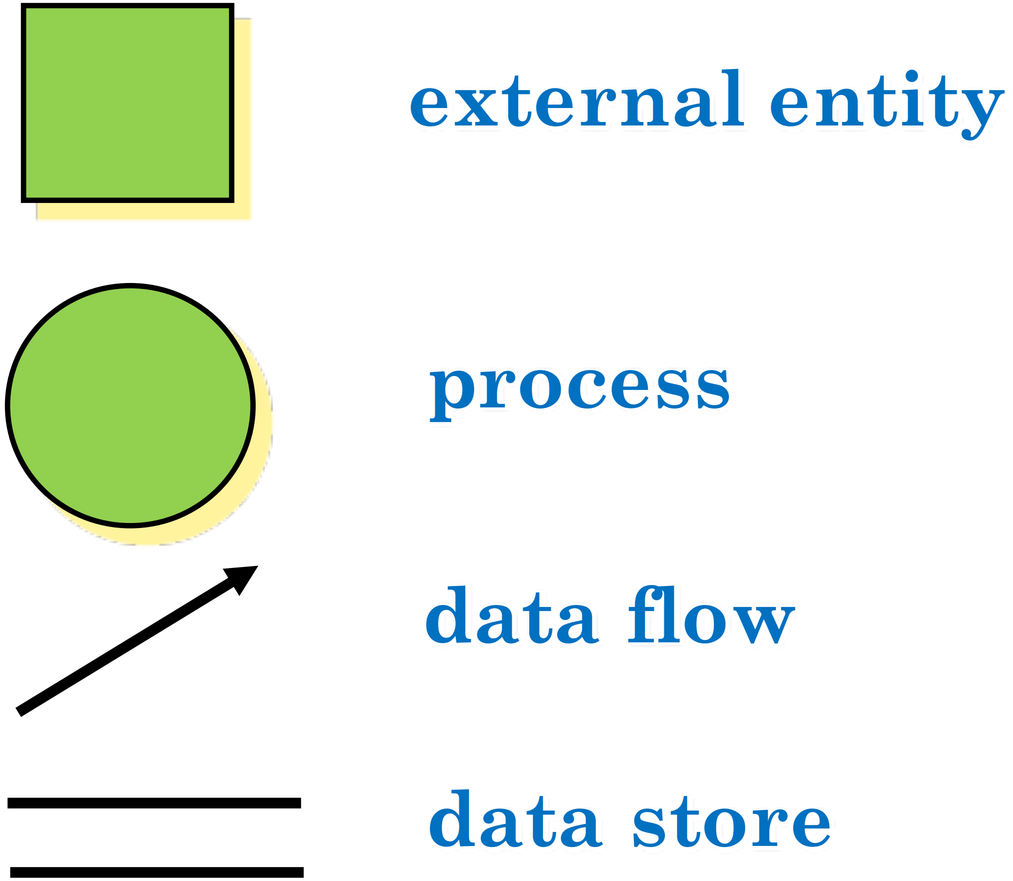
Every computer-based system is an information transform ....



Data Flow-Oriented Modeling

- One of the most widely used analysis notations
- Complement UML diagrams by providing flow-oriented analysis
- Data Flow Diagram (DFD)
  - Takes an input-process-output view of a system
    - Data objects: labeled arrows
    - Transformations (Process): circles (bubbles)
  - An overall description of the function to be developed
    - Contrast: scenario described from one actor's point of view

Flow Modeling Notation



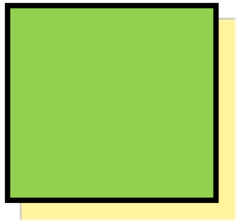
external entity

process

data flow

data store

External Entity



external entity

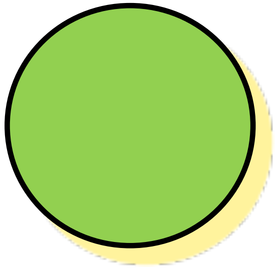
A producer or consumer of data

Examples: a person, a device, a sensor

Another example: computer-based system

Data must always originate somewhere and must always be sent to something

Process



process

A data transformer (changes input to output)

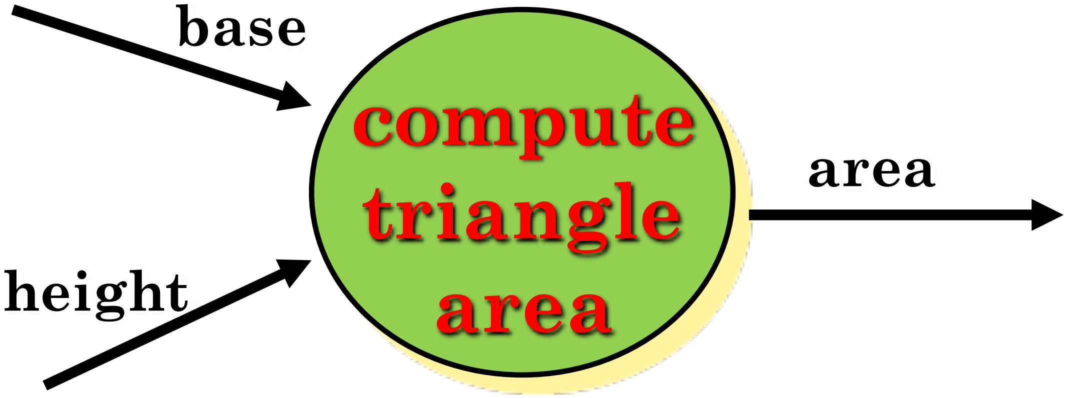
Examples: *compute taxes, determine area, format report, display graph*

Data must always be processed in some way to achieve system function

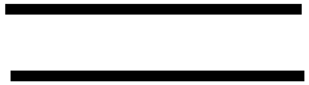
DATA FLOW

→ **data flow**

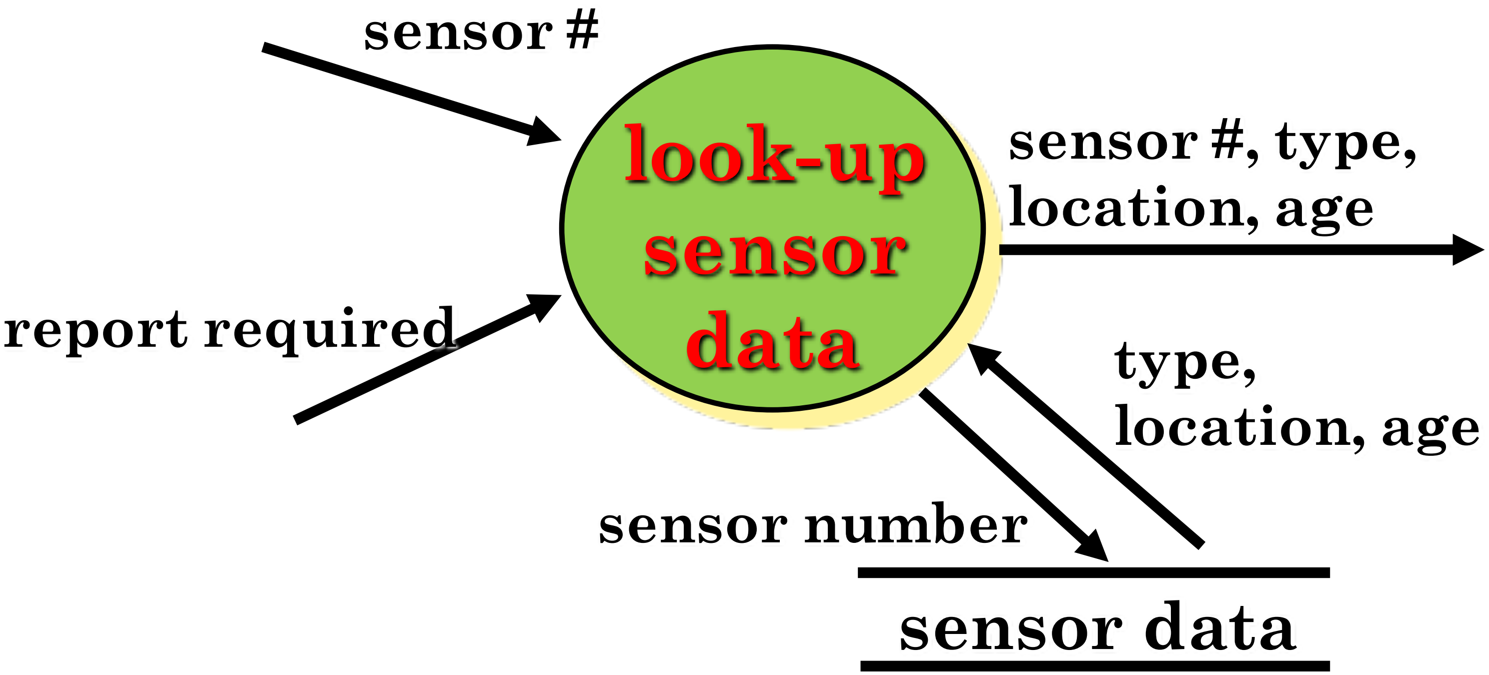
Data flows through a system, beginning as input and transformed into output.



DATA STORES

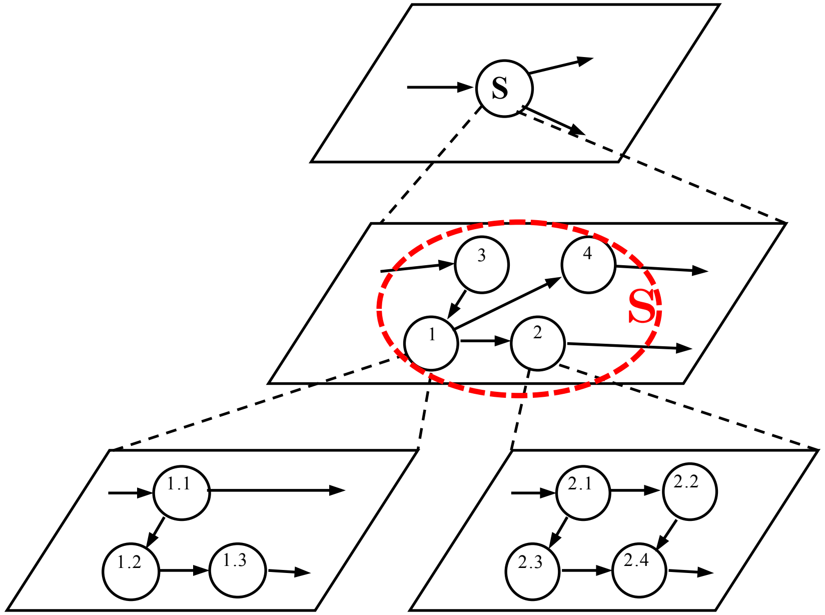


Data is often stored for later use.

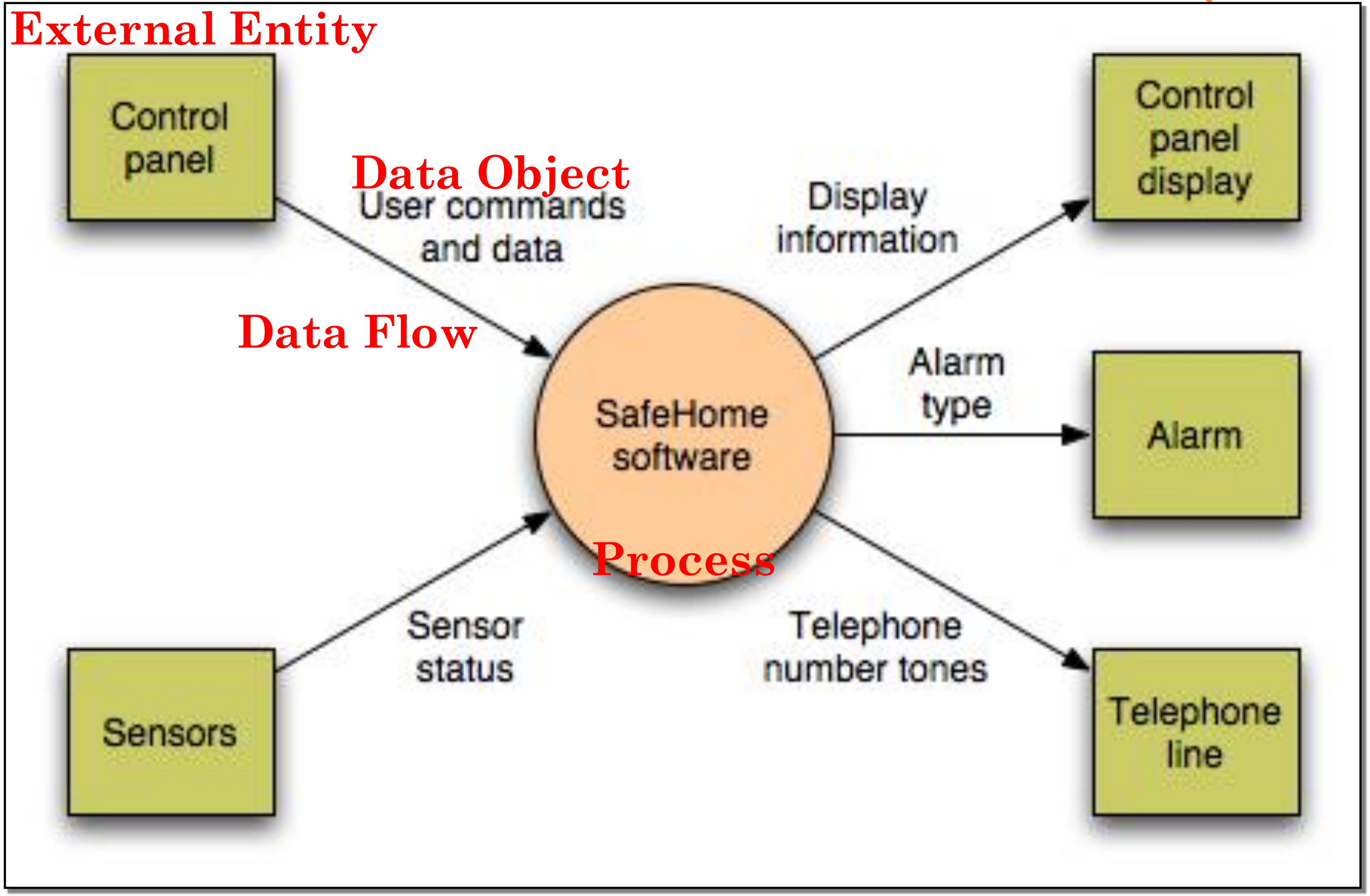


DFD: Hierarchical Refinement

- First data flow model (level 0 DFD or context diagram) represents the system as a whole
- Subsequent data flow diagrams refine the context diagram providing increasing details with each subsequent level

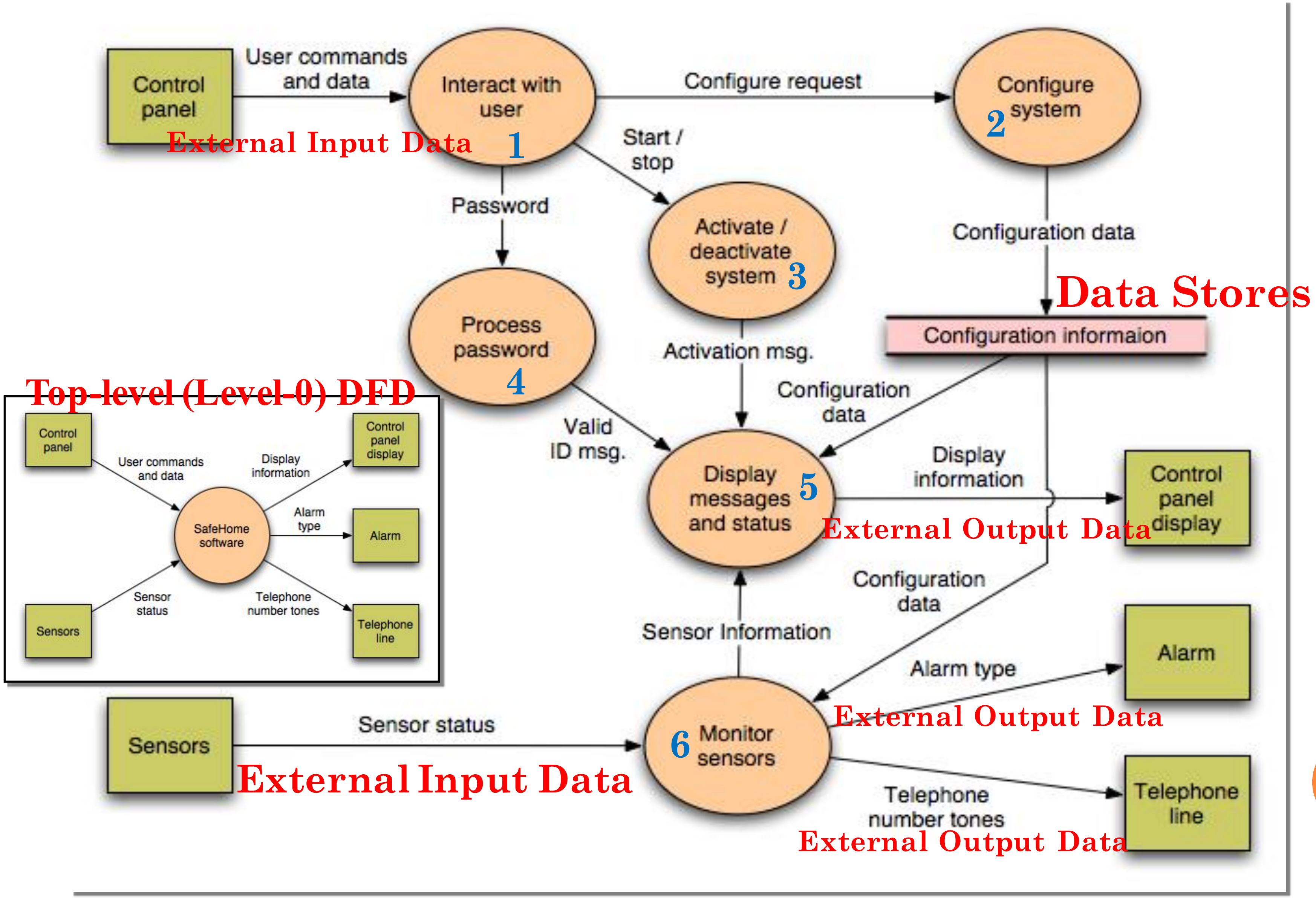


DATA FLOW DIAGRAM(DFD)



Top-level (Level-0) DFD for *SafeHome* security function

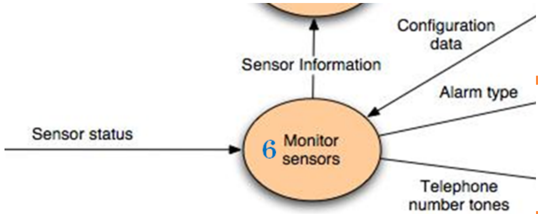
LEVEL-1 DFD


Control panel
User commands and data
Interact with user
1
External Input Data
Configure request
Configure system
2
Start / stop
Password
Activate / deactivate system
3
Configuration data
Data Stores
Configuration informaion
Process password
4
Activation msg.
Top-level (Level-0) DFD
Control panel
User commands and data
Display information
Control panel display
SafeHome software
Alarm type
Alarm
Sensors
Sensor status
Telephone number tones
Telephone line
Valid ID msg.
Configuration data
Display messages and status
5
Display information
Control panel display
External Output Data
Configuration data
Sensor Information
Alarm type
Alarm
External Output Data
Sensors
Sensor status
External Input Data
6
Monitor sensors
Telephone number tones
Telephone line
External Output Data

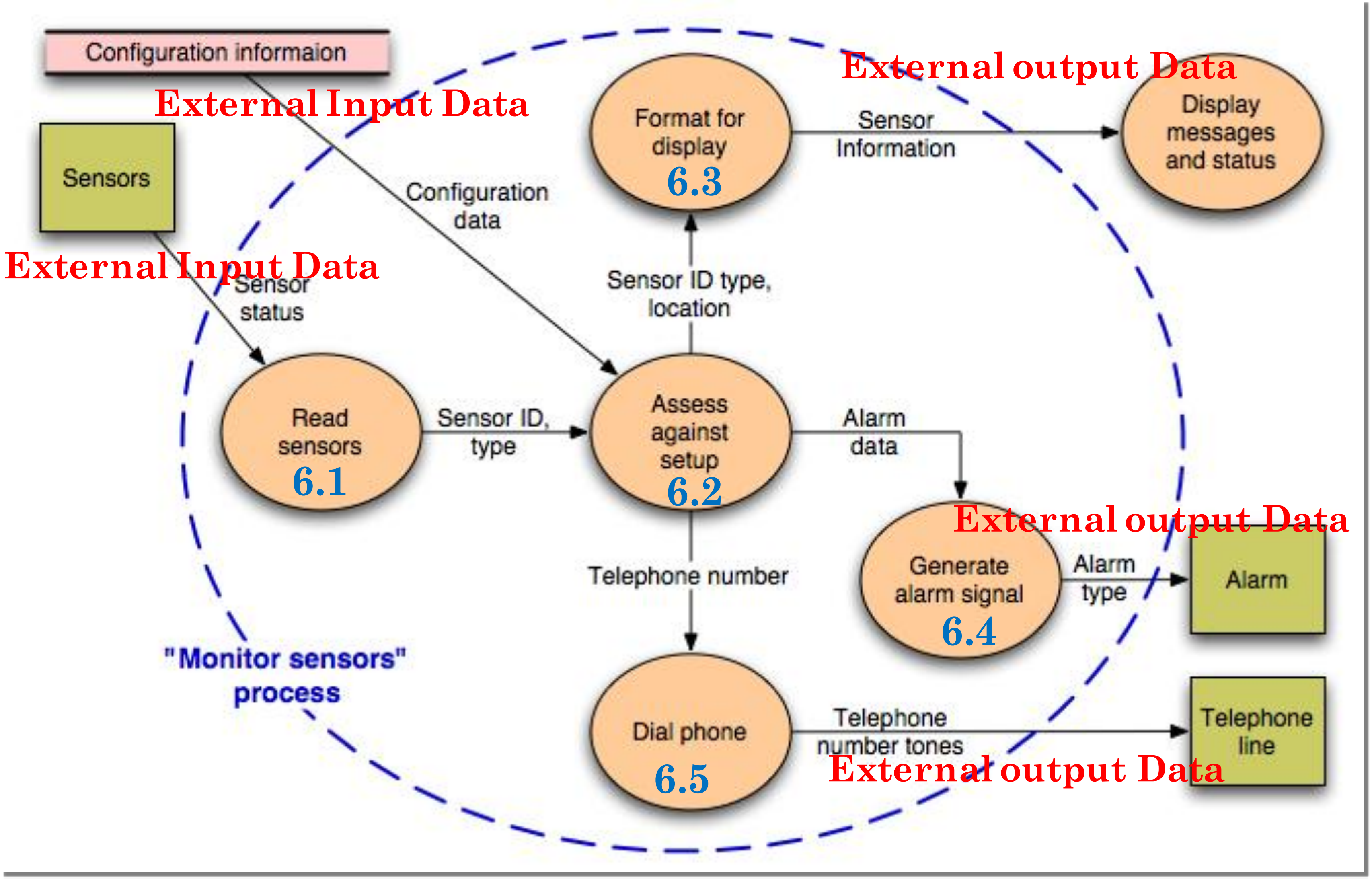
How to Refine?

- Grammatical parse on the narrative describing upper-level process (bubble)
  - Verbs: processes
  - Nouns: external entities, data or control objects, or data stores
  - Further associate nouns and verbs with one another
    - Data object and its attributes
    - Processes and its input/output data objects

The SafeHome security function ***enables*** the homeowner to ***configure*** the security system when it is ***installed***, ***monitors*** all sensors ***connected*** to the security system, and ***interacts*** with the homeowner through the Internet, a PC or a control panel…..



LEVEL-2 DFD



External Input Data

External Input Data

External output Data

External output Data

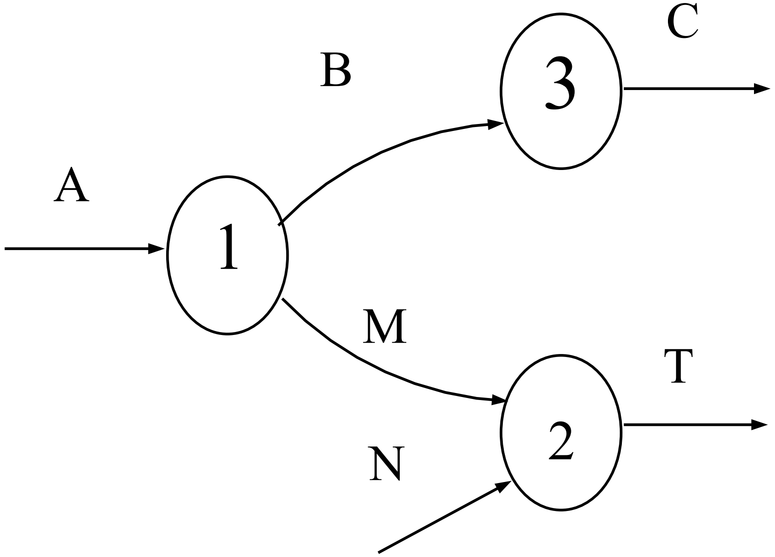
External output Data

Level-2 DFD that refines the *Monitor sensors* process (6)

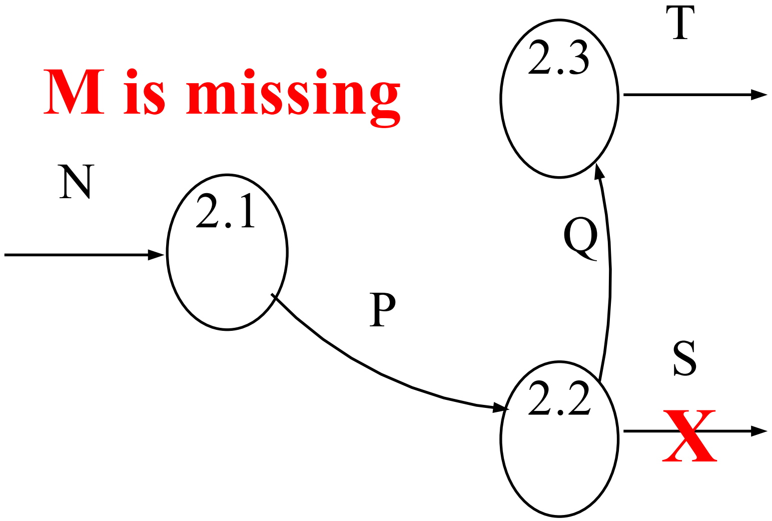
Consistency in DFD Refinement

- Information flow continuity
  - External input/output data flows should be consistent with those in the parent diagram
- Data store is local
  - Data stores can only be accessed by local processes in the same DFD
  - Related data flows can be referred in refined DFDs
- All the output data must be included in certain input data or can be produced by the process
- Each output data object cannot be named the same with any input data of the same process

Example of Violation



(a) Parent DFD


M is missing

(b) Refined DFD

FURTHER REFINEMENT…

- Further refinement
  - Level-2 DFDs for process 1-5
  - Level-3 DFDs for Level-2 processes
  - ……
- Refinement continues until each bubble performs a simple function
  - Would be easily implemented as a program component
  - The concept of "cohesion" can be used to assess the simplicity (discussed in design engineering)

DFD Guidelines

- Depict the system as single bubble in level 0
- Carefully note primary input and output
- Label all elements with meaningful names
- Refine by isolating candidate processes and their associated data objects and data stores
- Maintain information conformity between levels
- Refine one bubble at a time
- Don't show too much detail too early or represents procedural aspects of the software
  (7±2 processes in each diagram)

DATA DICTIONARY

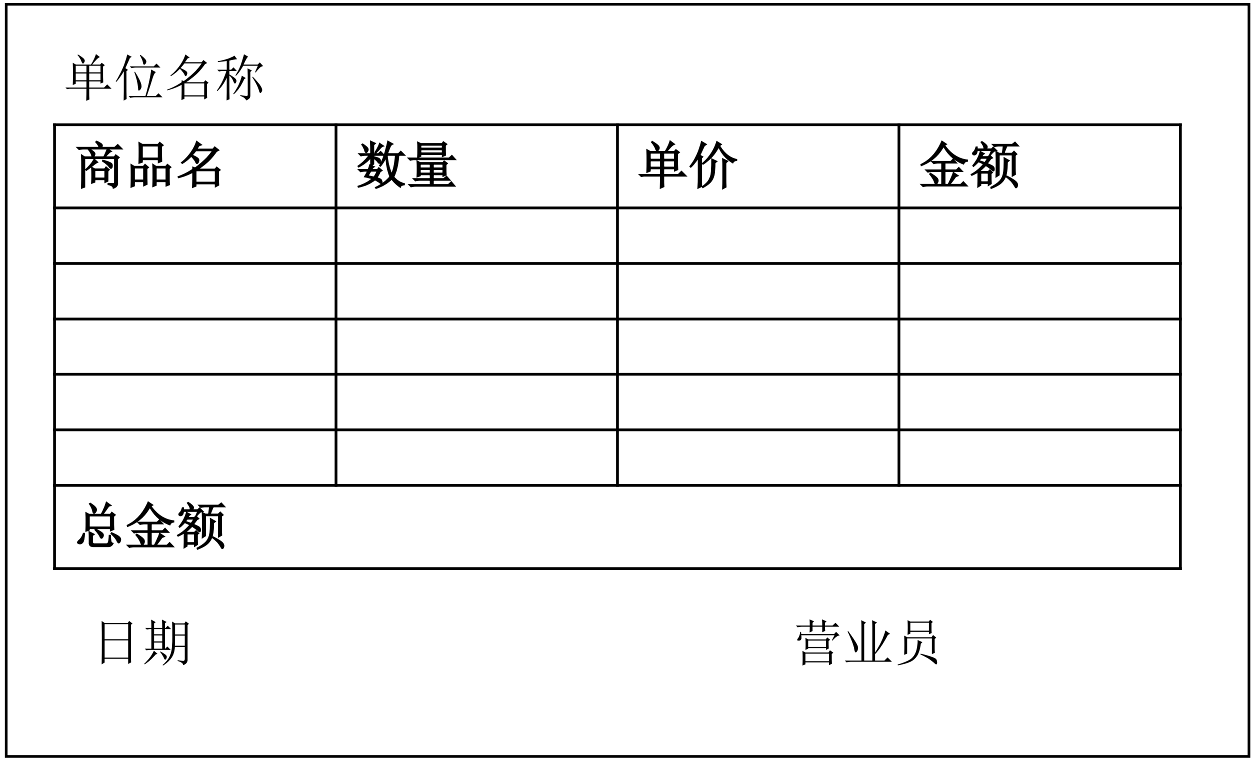
- Data Dictionary
  - Structural definition for data items
  - Includes descriptions for
    - Data flow
    - Data store (file)
    - Data items composing data flow or file
    - External entities
    - Process specification

Structural Definition for Data Objects

符号	名称	举例
＝	定义为	x＝… 表示x由…组成
＋	与	a＋b　表示a和b
［…，　…］	或	［a，　b］表示a或b
［…|…］	或	［a|b］表示a或b
{…}	重复	{a}　表示a重复0或多次
$\{\ldots\}_m^n$	重复	$\{a\}_3^8$表示a重复3到8次
(…)	可选	(a)　表示a重复0或1次
″…″	基本数据元素	″a″　表示a是基本数据

DATA FLOW DESCRIPTION EXAMPLE

发票＝单位名称＋｛商品名＋数量＋单价＋金额｝
　　　＋总金额＋日期＋(营业员)

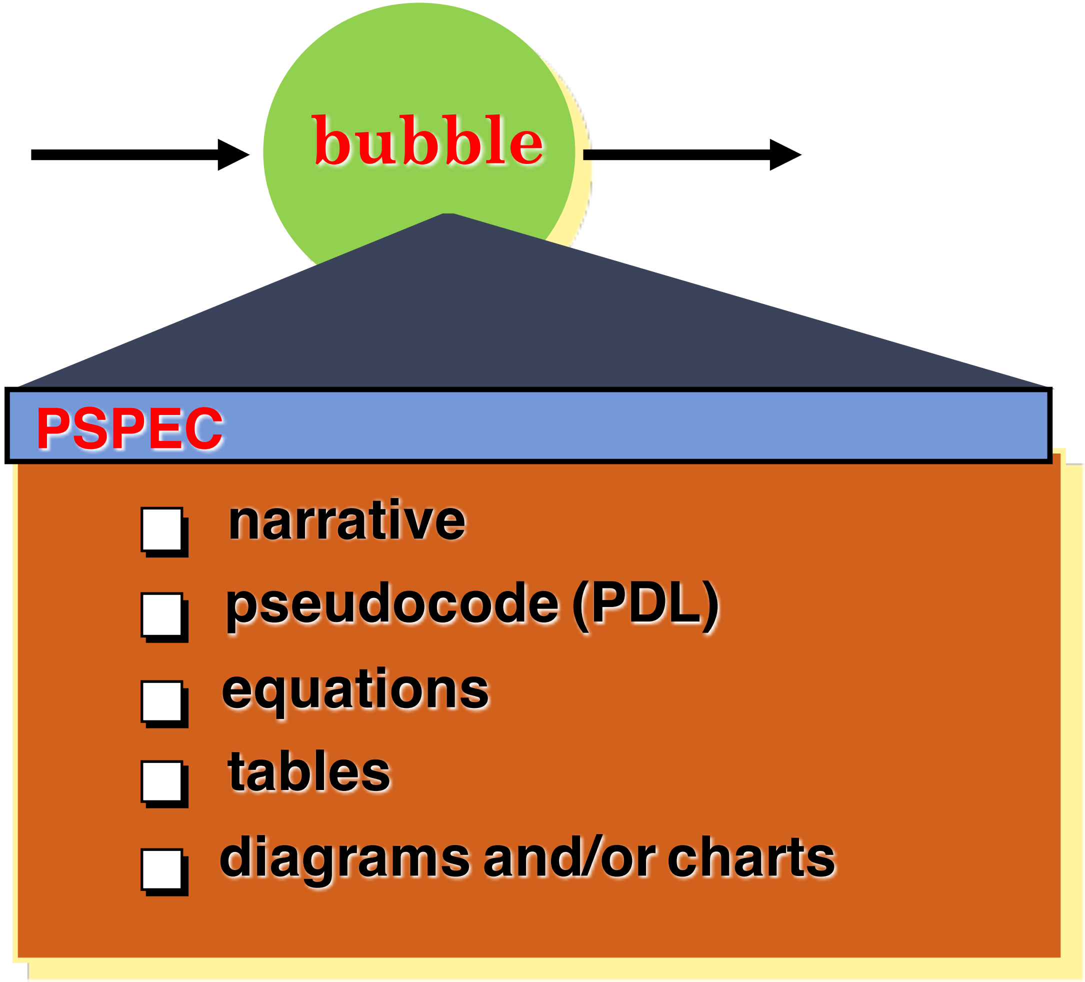


单位名称

商品名	数量	单价	金额
总金额			

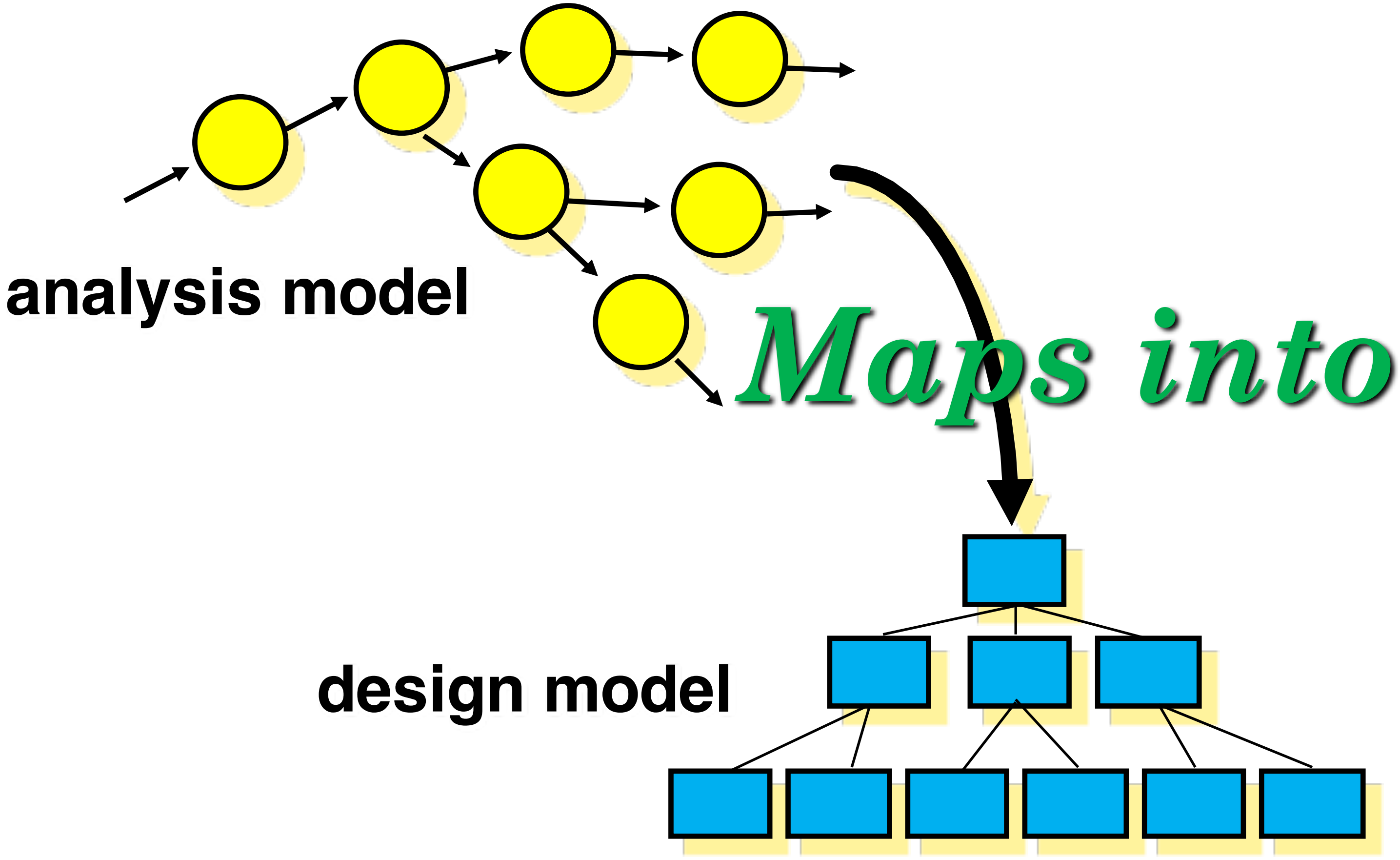
日期　　　　　　　　　　营业员

Process Specification (PSPEC)



- □ narrative
- □ pseudocode (PDL)
- □ equations
- □ tables
- □ diagrams and/or charts

DFDs: A Look Ahead



Control Flow Model

- Control information cannot be specified in data flow models
  - Applications “driven” by events rather than data
  - Applications produce control information rather than reports or displays
  - Process information with heavy concern for time and performance

EVENTS

- Represents "events" and the processes that manage events
- An "event" is a Boolean condition that can be ascertained by:
  - listing all sensors that are "read" by the software
  - listing all interrupt conditions
  - listing all "switches" that are actuated by an operator
  - listing all data conditions
  - recalling the noun/verb parse that was applied to the processing narrative, review all "control items" as possible CSPEC inputs/outputs

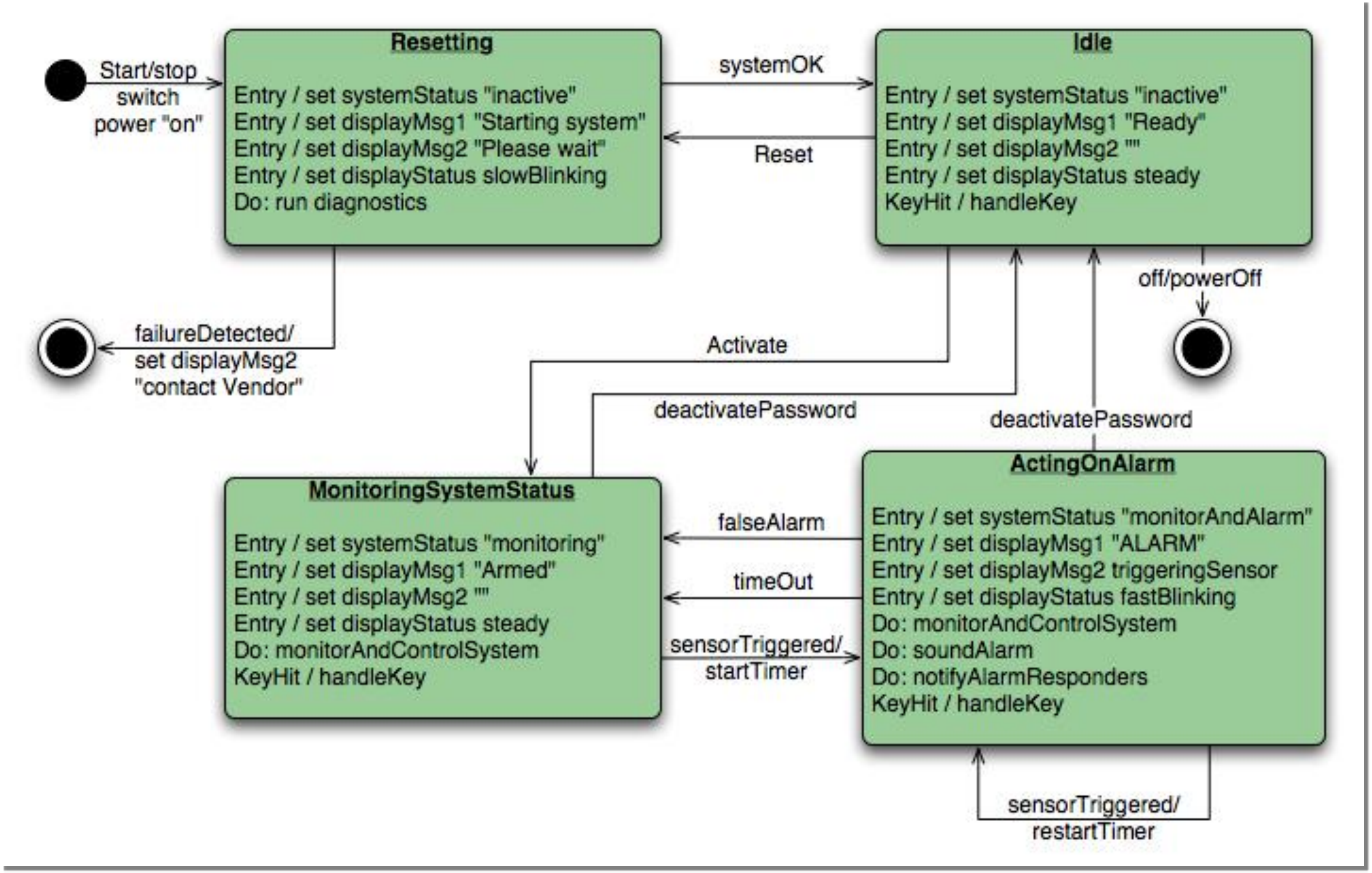
Control Specification (CSPEC)

The CSPEC can be:

- state diagram (sequential spec)
- state transition table
- decision tables
- activation tables

combinatorial spec

CONTROL FLOW DIAGRAM


Start/stop switch power "on"
Resetting
Entry / set systemStatus "inactive"
Entry / set displayMsg1 "Starting system"
Entry / set displayMsg2 "Please wait"
Entry / set displayStatus slowBlinking
Do: run diagnostics
systemOK
Reset
Idle
Entry / set systemStatus "inactive"
Entry / set displayMsg1 "Ready"
Entry / set displayMsg2 ""
Entry / set displayStatus steady
KeyHit / handleKey
failureDetected/ set displayMsg2 "contact Vendor"
off/powerOff
Activate
deactivatePassword
deactivatePassword
MonitoringSystemStatus
Entry / set systemStatus "monitoring"
Entry / set displayMsg1 "Armed"
Entry / set displayMsg2 ""
Entry / set displayStatus steady
Do: monitorAndControlSystem
KeyHit / handleKey
falseAlarm
timeOut
sensorTriggered/ startTimer
ActingOnAlarm
Entry / set systemStatus "monitorAndAlarm"
Entry / set displayMsg1 "ALARM"
Entry / set displayMsg2 triggeringSensor
Entry / set displayStatus fastBlinking
Do: monitorAndControlSystem
Do: soundAlarm
Do: notifyAlarmResponders
KeyHit / handleKey
sensorTriggered/ restartTimer

State machine diagram for *SafeHome* security function

Behavioral Modeling

BEHAVIORAL MODELING

- Behavioral Model indicates how software will respond to external events or stimuli
- Behavioral Modeling Steps
  - Evaluate use cases to understand interaction sequences
  - Identify events that drive interactions and understand how these events relate to specific objects
  - Create a sequence for each use case
  - Build a state diagram for the system
  - Review behavioral models for accuracy and consistency

Identifying Events

- Event: not the information been exchanged, but rather the fact that information has been exchanged
  - Example: password entered Vs. entered password
- Identifying Events: use cases are examined for ***points of information exchange***
  - Example:
    The homeowner uses the keypad to key in a four-digit password. The password is compared with the valid password stored in the system. If the password in incorrect, the control panel will beep once and reset itself for additional input. If the password is correct, the control panel awaits further action.

STATE REPRESENTATIONS

- In the context of behavioral modeling, two different characterizations of states must be considered:
  - the state of each class as the system performs its function and
  - the state of the system as observed from the outside as the system performs its function
- The state of a class takes on both passive and active characteristics [CHA93].
  - 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.

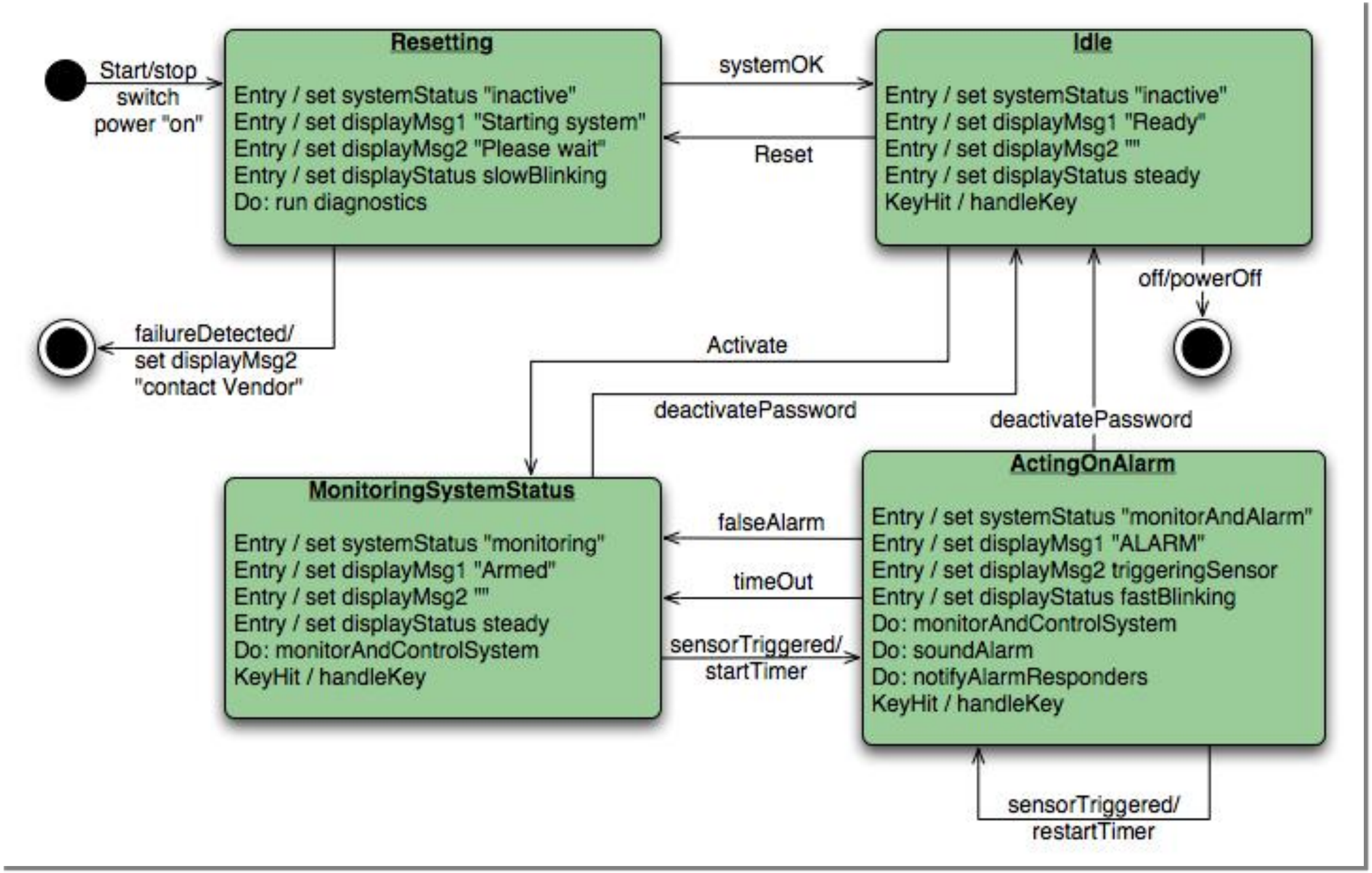
ELEMENTS OF BEHAVIORAL MODEL

- state—a set of observable circumstance that characterizes the behavior of a system at a given time
- state transition—the movement from one state to another
- event—an occurrence that causes the system to exhibit some predictable form of behavior
- action—process that occurs as a consequence of making a transition

BEHAVIORAL MODELING

- make a list of the different states of a system (How does the system behave?)
- indicate how the system makes a transition from one state to another (How does the system change state?)
  - indicate event
  - indicate action
- draw a state diagram or a sequence diagram

State Diagram



Resetting

Entry / set systemStatus "inactive"
Entry / set displayMsg1 "Starting system"
Entry / set displayMsg2 "Please wait"
Entry / set displayStatus slowBlinking
Do: run diagnostics

Idle

Entry / set systemStatus "inactive"
Entry / set displayMsg1 "Ready"
Entry / set displayMsg2 ""
Entry / set displayStatus steady
KeyHit / handleKey

MonitoringSystemStatus

Entry / set systemStatus "monitoring"
Entry / set displayMsg1 "Armed"
Entry / set displayMsg2 ""
Entry / set displayStatus steady
Do: monitorAndControlSystem
KeyHit / handleKey

ActingOnAlarm

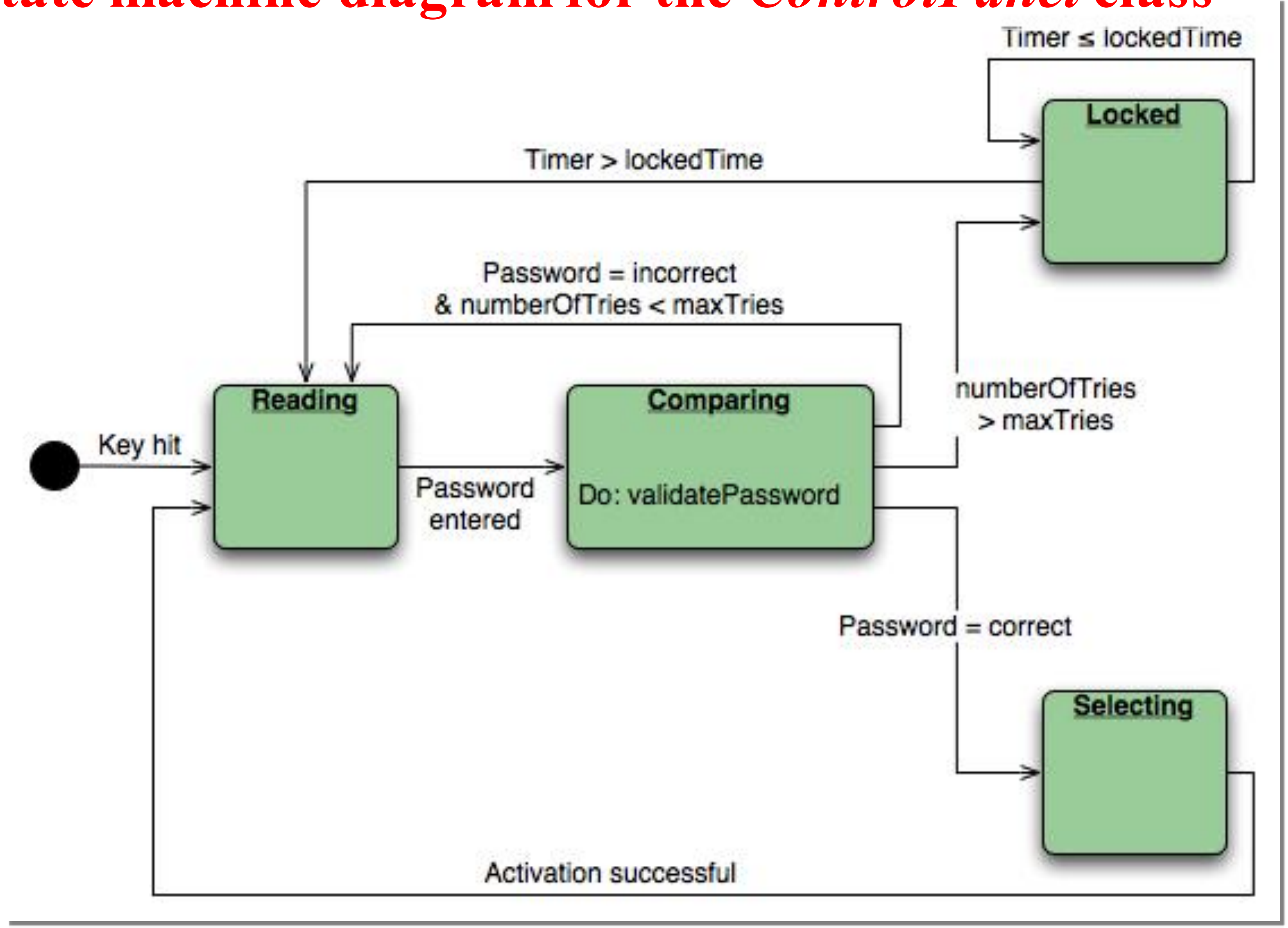
Entry / set systemStatus "monitorAndAlarm"
Entry / set displayMsg1 "ALARM"
Entry / set displayMsg2 triggeringSensor
Entry / set displayStatus fastBlinking
Do: monitorAndControlSystem
Do: soundAlarm
Do: notifyAlarmResponders
KeyHit / handleKey

Start/stop switch power "on"
systemOK
Reset
off/powerOff
failureDetected/ set displayMsg2 "contact Vendor"
Activate
deactivatePassword
deactivatePassword
falseAlarm
timeOut
sensorTriggered/ startTimer
sensorTriggered/ restartTimer

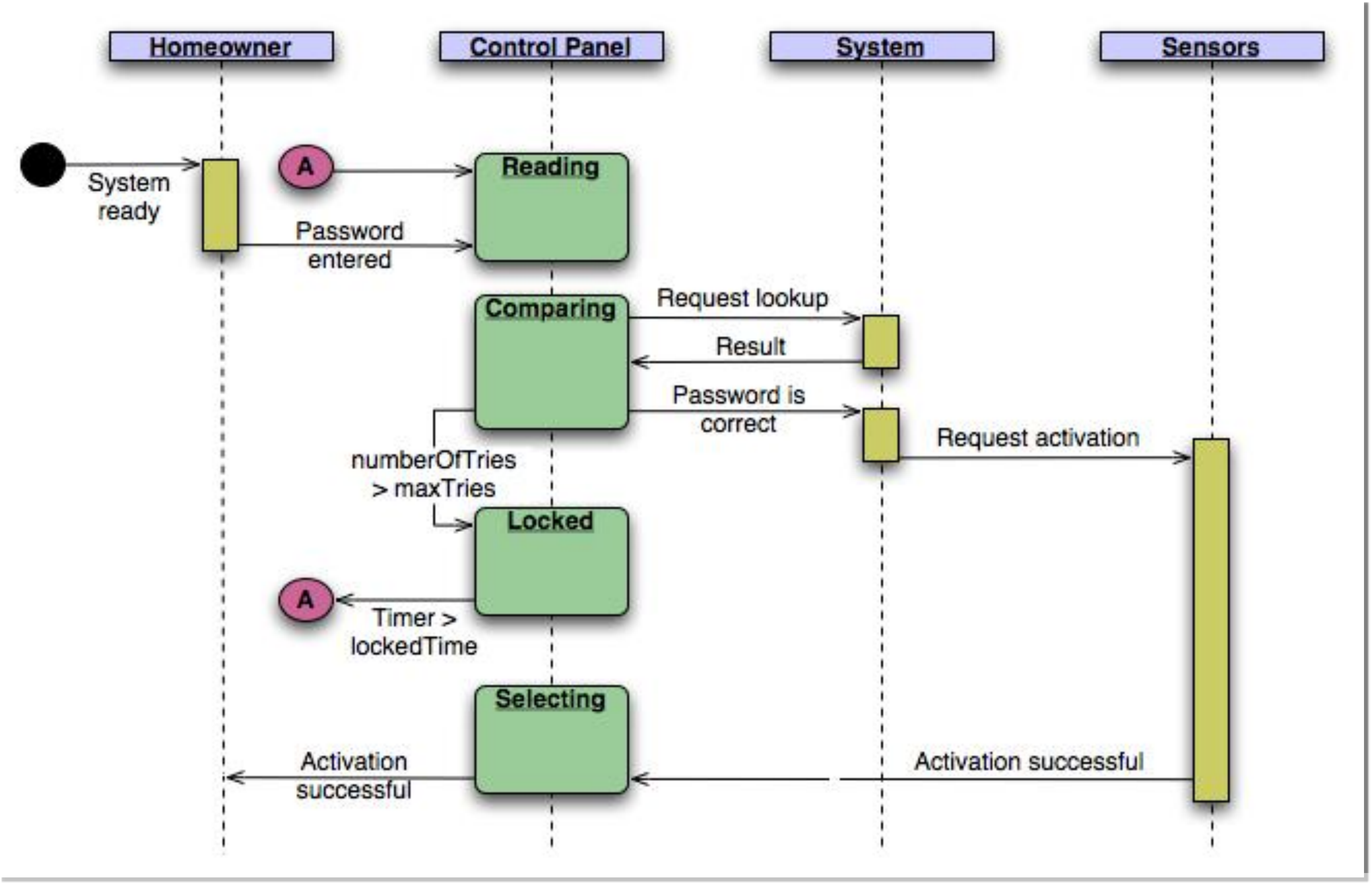
State machine diagram for *SafeHome* security function

State Diagram

State machine diagram for the *ControlPanel* class



SEQUENCE DIAGRAM



Sequence diagram (partial) for the *SafeHome* security function

Patterns for Requirements Modeling

Patterns for Requirements Modeling

- Software patterns are a mechanism for capturing domain knowledge in a way that allows it to be reapplied when a new problem is encountered
  - domain knowledge can be applied to a new problem within the same application domain
  - the domain knowledge captured by a pattern can be applied by analogy to a completely different application domain
- The original author of an analysis pattern does not "create" the pattern, but rather, discovers it as requirements engineering work is being conducted
- Once the pattern has been discovered, it is documented

DISCOVERING ANALYSIS PATTERNS

- The most basic element in the description of a requirements model is the use case
- A coherent set of use cases may serve as the basis for discovering one or more analysis patterns
- A *semantic analysis pattern* (SAP) "is a pattern that describes a small set of coherent use cases that together describe a basic generic application." [Fer00]

AN EXAMPLE

- 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. The control software superimposes a variety of distance and orientation lines on the dashboard display so that the vehicle operator can maintain orientation as the vehicle moves in reverse. The control software also monitors a proximity sensor to determine whether an object is inside 10 feet of the rear of the vehicle. It will automatically break the vehicle if the proximity sensor indicates an object within 3 feet of the rear of the vehicle.

AN EXAMPLE

- This use case implies a variety of functionality that would be refined and elaborated (into a coherent set of use cases) during requirements gathering and modeling
- Regardless of how much elaboration is accomplished, the use case(s) suggest(s) a simple, yet widely applicable SAP—the software-based monitoring and control of sensors and actuators in a physical system
- In this case, the "sensors" provide information about proximity and video information. The "actuator" is the breaking system of the vehicle (invoked if an object is very close to the vehicle)
- But in a more general case, a widely applicable pattern is discovered --> **Actuator-Sensor**

ACTUATOR-SENSOR PATTERN-I

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. Although many of the sensors and actuators look quite different, their behavior is similar enough to structure them into a pattern. The pattern shows how to specify the sensors and actuators for a system, including attributes and operations. The *Actuator-Sensor* pattern uses a *pull* mechanism (explicit request for information) for **PassiveSensors** and a *push* mechanism (broadcast of information) for the **ActiveSensors**.

Constraints:

Each passive sensor must have some method to read sensor input and attributes that represent the sensor value.

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 detect malfunctions.

Each actuator must have some method to invoke the appropriate response determined by the **ComputingComponent**.

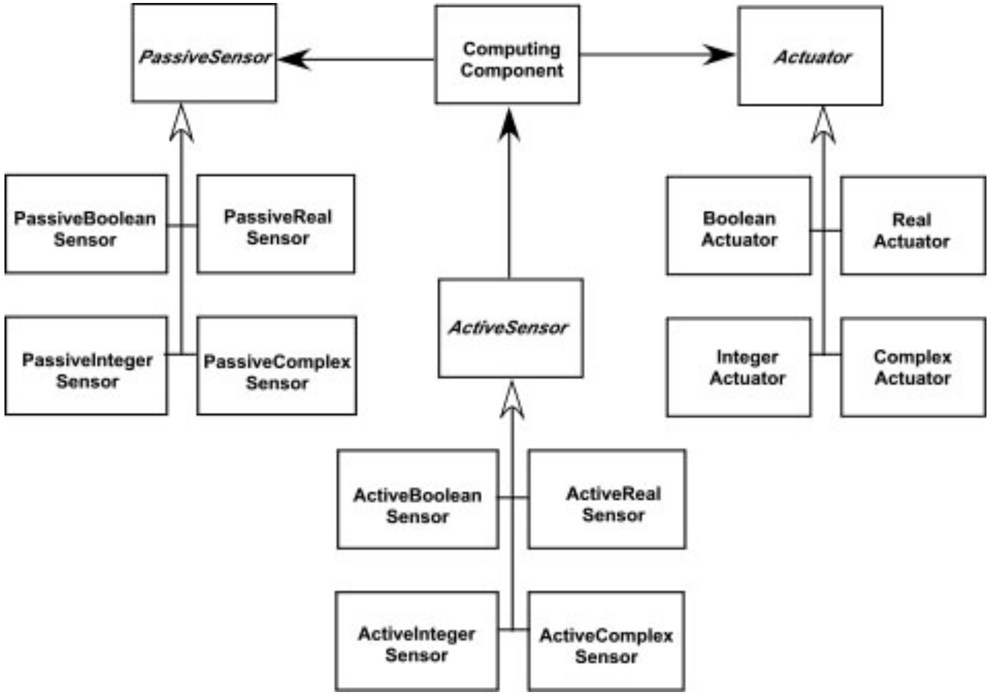
Each sensor and actuator should have a function implemented to check its own operation state.

Each sensor and actuator should be able to test the validity of the values received or sent and set its operation state if the values are outside of the specifications.

ACTUATOR-SENSOR PATTERN-II

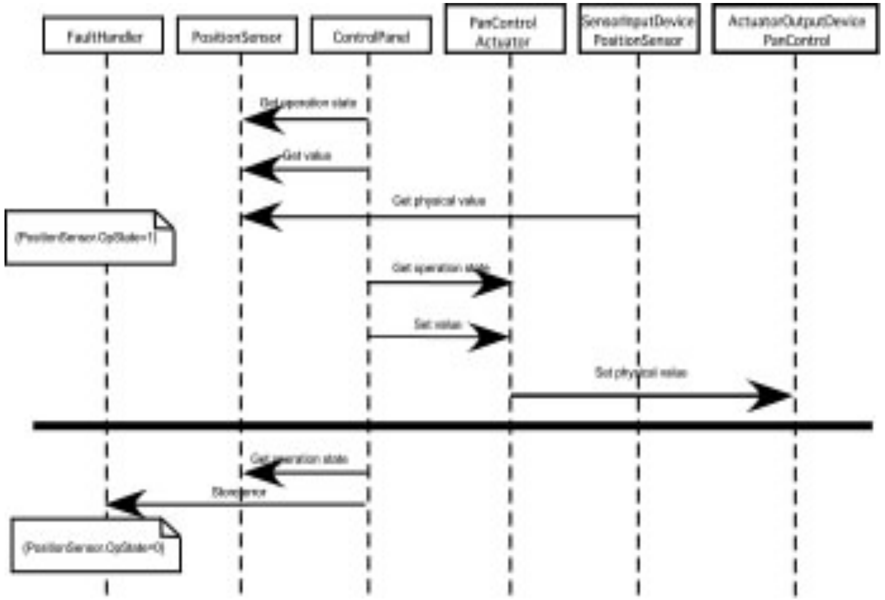
Applicability: Useful in any system in which multiple sensors and actuators are present.

Structure: A UML class diagram for the *Actuator-Sensor* Pattern is shown in Figure 7.8. **Actuator**, **PassiveSensor** and **ActiveSensor** are abstract classes and denoted in italics. There are four different types of sensors and actuators in this pattern. The Boolean, integer, and real classes represent the most common types of sensors and actuators. The complex classes are sensors or actuators that use values that cannot be easily represented in terms of primitive data types, such as a radar device. Nonetheless, these devices should still inherit the interface from the abstract classes since they should have basic functionalities such as querying the operation states.



ACTUATOR-SENSOR PATTERN-III

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. Here, the **ControlPanel** queries a sensor (a passive position sensor) and an actuator (pan control) to check the operation state for diagnostic purposes before reading or setting a value. The messages *Set Physical Value* and *Get Physical Value* are not messages between objects. Instead, they describe the interaction between the physical devices of the system and their software counterparts. In the lower part of the diagram, below the horizontal line, the **PositionSensor** reports that the operation state is zero. The **ComputingComponent** then sends the error code for a position sensor failure to the **FaultHandler** that will decide how this error affects the system and what actions are required. it gets the data from the sensors and computes the required response for the actuators.



Requirements Modeling for WebApps

Requirements Modeling for WebApps

- Content Analysis. The full spectrum of content to be provided by the WebApp is identified, including text, graphics and images, video, and audio data. Data modeling can be used to identify and describe each of the data objects.
- Interaction Analysis. The manner in which the user interacts with the WebApp is described in detail. Use-cases can be developed to provide detailed descriptions of this interaction.
- Functional Analysis. The usage scenarios (use-cases) created as part of interaction analysis define the operations that will be applied to WebApp content and imply other processing functions. All operations and functions are described in detail.
- Configuration Analysis. The environment and infrastructure in which the WebApp resides are described in detail.

WHEN DO WE PERFORM ANALYSIS?

- In some cases of WebApps development, analysis and design merge
- However, an explicit analysis activity occurs when …
  - the WebApp to be built is large and/or complex
  - the number of stakeholders is large
  - the number of Web engineers and other contributors is large
  - the goals and objectives (determined during formulation) for the WebApp will effect the business' bottom line
  - the success of the WebApp will have a strong bearing on the success of the business

THE CONTENT MODEL

- Content objects are extracted from use-cases
  - examine the scenario description for direct and indirect references to content
- Attributes of each content object are identified
- The relationships among content objects and/or the hierarchy of content maintained by a WebApp
  - Relationships—entity-relationship diagram or UML
  - Hierarchy—data tree or UML

Data Tree

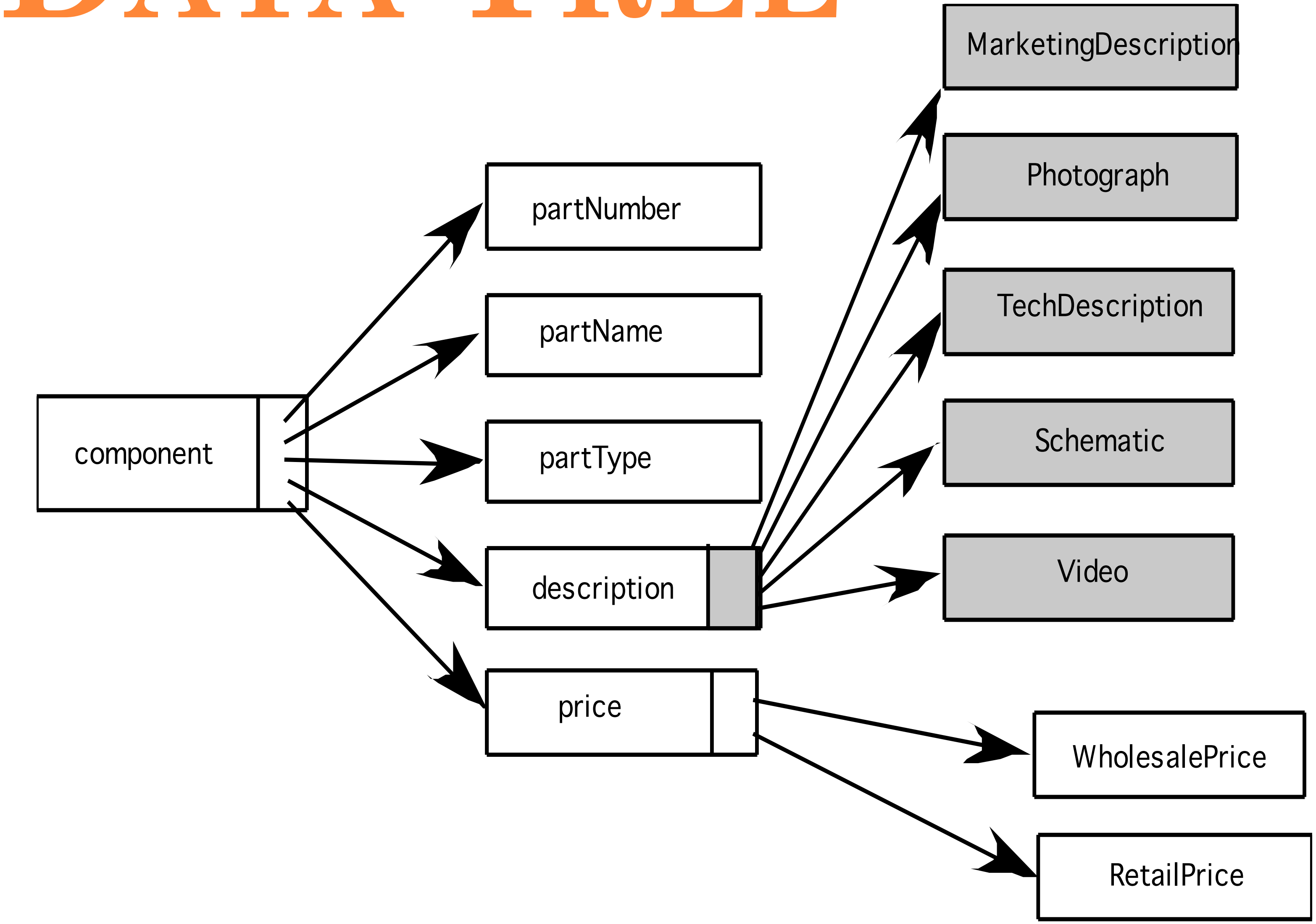


Figure 18.3 Data tree for a *SafeHome* component

THE INTERACTION MODEL

- Composed of four elements:
  - use-cases
  - sequence diagrams
  - state diagrams
  - a user interface prototype

SEQUENCE DIAGRAM

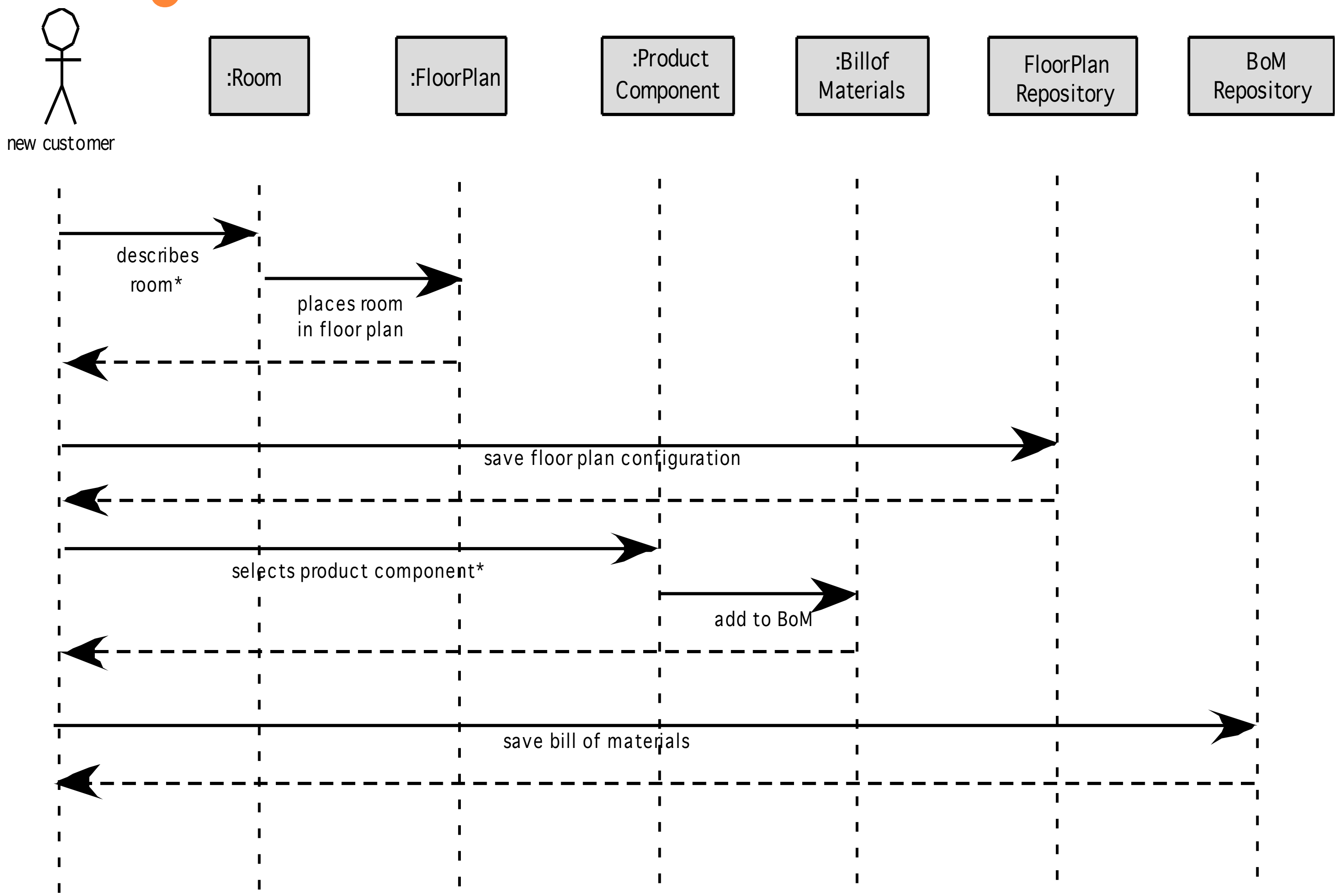
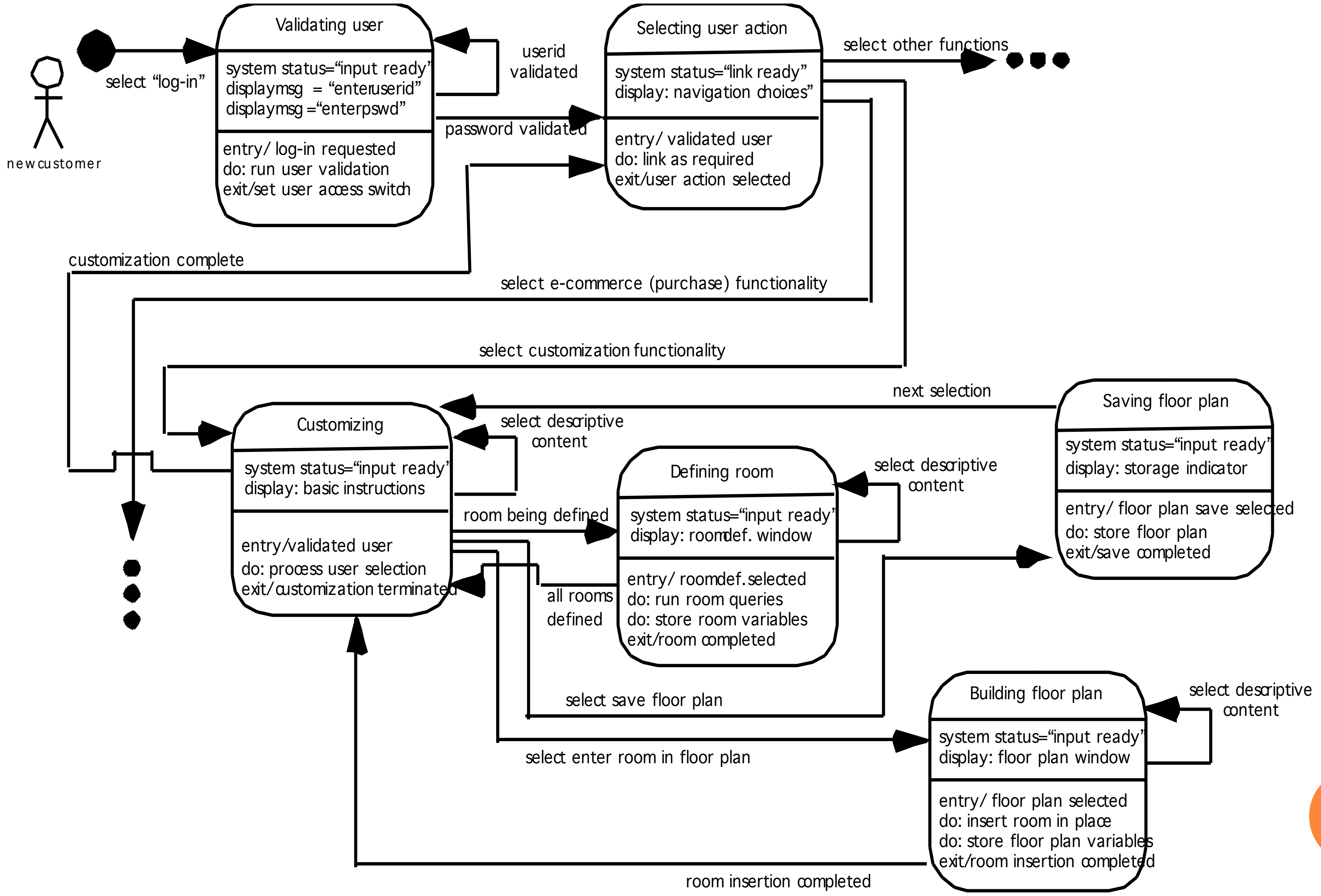


Figure 18.5 Sequence diagram for use-case:*select SafeHome components*

State Diagram



Validating user
system status="input ready"
displaymsg = "enteruserid"
displaymsg ="enterpswd"
entry/ log-in requested
do: run user validation
exit/set user access switch
select "log-in"
new customer
userid validated
password validated
Selecting user action
system status="link ready"
display: navigation choices"
entry/ validated user
do: link as required
exit/user action selected
select other functions
customization complete
select e-commerce (purchase) functionality
select customization functionality
next selection
Customizing
system status="input ready"
display: basic instructions
entry/validated user
do: process user selection
exit/customization terminated
select descriptive content
room being defined
all rooms defined
Defining room
system status="input ready"
display: roomdef. window
entry/ roomdef. selected
do: run room queries
do: store room variables
exit/room completed
select descriptive content
Saving floor plan
system status="input ready"
display: storage indicator
entry/ floor plan save selected
do: store floor plan
exit/save completed
select save floor plan
select enter room in floor plan
Building floor plan
system status="input ready"
display: floor plan window
entry/ floor plan selected
do: insert room in place
do: store floor plan variables
exit/room insertion completed
select descriptive content
room insertion completed

Figure 18.6 Partial state diagram for **new customer** interaction

Software School, Fudan University
Spring Semester, 2016

THE FUNCTIONAL MODEL

- The functional model addresses two processing elements of the WebApp
  - user observable functionality that is delivered by the WebApp to end-users
  - the operations contained within analysis classes that implement behaviors associated with the class.
- An activity diagram can be used to represent processing flow

ACTIVITY DIAGRAM

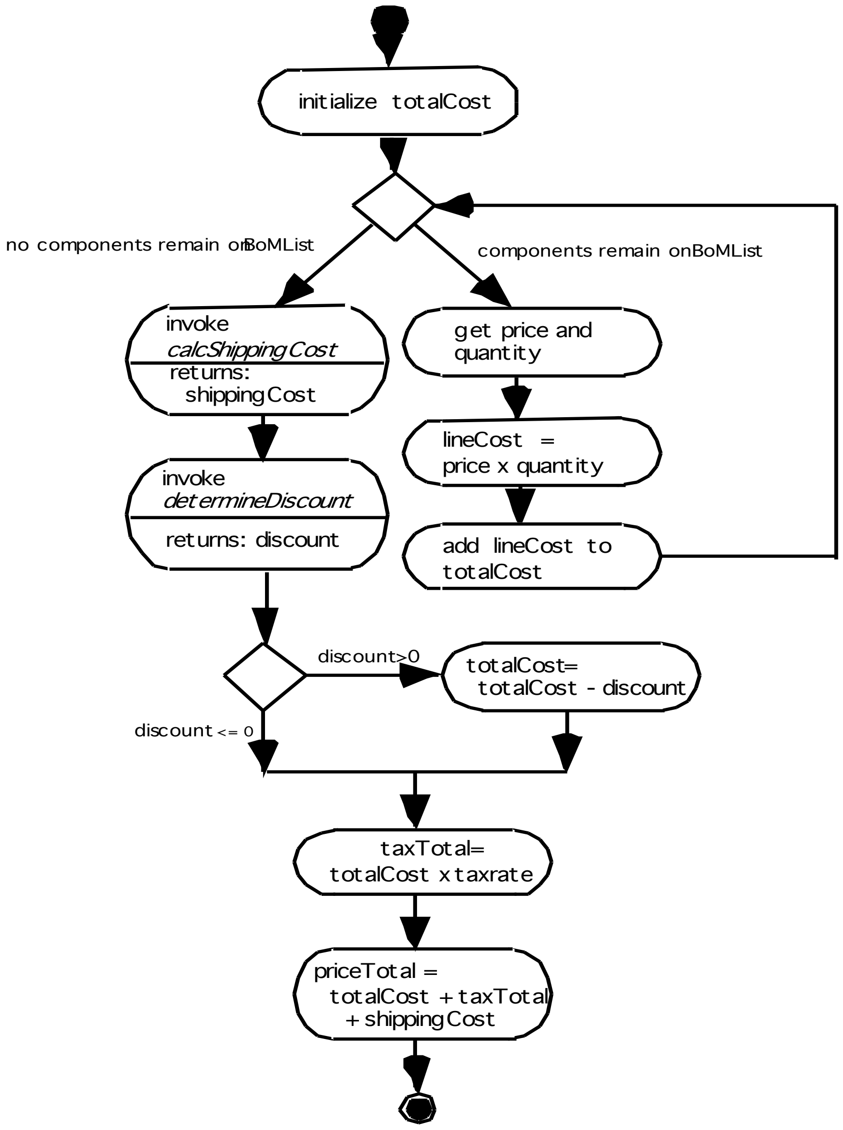


Figure 18.7 Activity diagram for *computePrice()* operatic

Software School, Fudan University
Spring Semester, 2016

THE CONFIGURATION MODEL

Server-side

- Server hardware and operating system environment must be specified
- Interoperability considerations on the server-side must be considered
- Appropriate interfaces, communication protocols and related collaborative information must be specified

Client-side

- Browser configuration issues must be identified
- Testing requirements should be defined

NAVIGATION MODELING-I

- Should certain elements be easier to reach (require fewer navigation steps) than others? What is the priority for presentation?
- Should certain elements be emphasized to force users to navigate in their direction?
- How should navigation errors be handled?
- 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 for users?

NAVIGATION MODELING-II

- Should a full navigation map or menu (as opposed to a single "back" link or directed pointer) be available at every point in a user's interaction?
- Should navigation design be driven by the most commonly expected user behaviors or by the perceived importance of the defined WebApp elements?
- Can a user "store" his previous navigation through the WebApp to expedite future usage?
- For which user category should optimal navigation be designed?
- How should links external to the WebApp be handled? overlaying the existing browser window? as a new browser window? as a separate frame?

End of Chapter 7