Software Engineering

Presenting by:
B.Pranalini

Unit 3 Part 2: Architectural Design

Topics

- Software Architecture
- Architectural Genres
- Architectural Styles
- Architectural Design
- Architectural Mapping using Data Flow

SOFTWARE ARCHITECTURE

WHAT IS ARCHITECTURE?

The architecture is not the operational software. Rather, it is a representation that enables a software engineer to:

(1) analyze the effectiveness of the design in meeting its stated requirements,

(2) consider architectural alternatives at a stage when making design changes is still relatively easy, and

(3) reduce the risks associated with the construction of the software.

Why is Architecture Important?

- Representations of software architecture are an enabler for communication between all parties (stakeholders) interested in the development of a computer-based system.
- The architecture highlights early design decisions that will have a profound impact on all software engineering work that follows and, as important, on the ultimate success of the system as an operational entity.
- Architecture "constitutes a relatively small, intellectually graspable mode of how the system is structured and how its components work together" [BAS03].

Architectural Descriptions

- The IEEE Computer Society has proposed IEEE-Std-1471-2000, *Recommended Practice for Architectural Description of Software-Intensive System,* [IEE00]
  - to establish a conceptual framework and vocabulary for use during the design of software architecture,
  - to provide detailed guidelines for representing an architectural description, and
  - to encourage sound architectural design practices.
- The IEEE Standard defines an *architectural description* (AD) as a "a collection of products to document an architecture."
  - The description itself is represented using multiple views, where each *view* is "a representation of a whole system from the perspective of a related set of [stakeholder] concerns."

Architectural Decisions

- Each view developed as part of an architectural description addresses a specific stakeholder concern.
- To develop each view (and the architectural description as a whole) the system architect considers a variety of alternatives and ultimately decides on the specific architectural features that best meet the concern
- Therefore, architectural decisions themselves can be considered to be one view of the architecture.

Architectural Genres

- *Genre* implies a specific category within the overall software domain.
- Within each category, you encounter a number of subcategories.
  - For example, within the genre of *buildings*, you would encounter the following general styles: houses, condos, apartment buildings, office buildings, industrial building, warehouses, and so on.
  - Within each general style, more specific styles might apply. Each style would have a structure that can be described using a set of predictable patterns.

Architectural Styles-A Brief Taxonomy of Architectural Styles

Each style describes a system category that encompasses:

(1) a **set of components** (e.g., a database, computational modules) that perform a function required by a system,

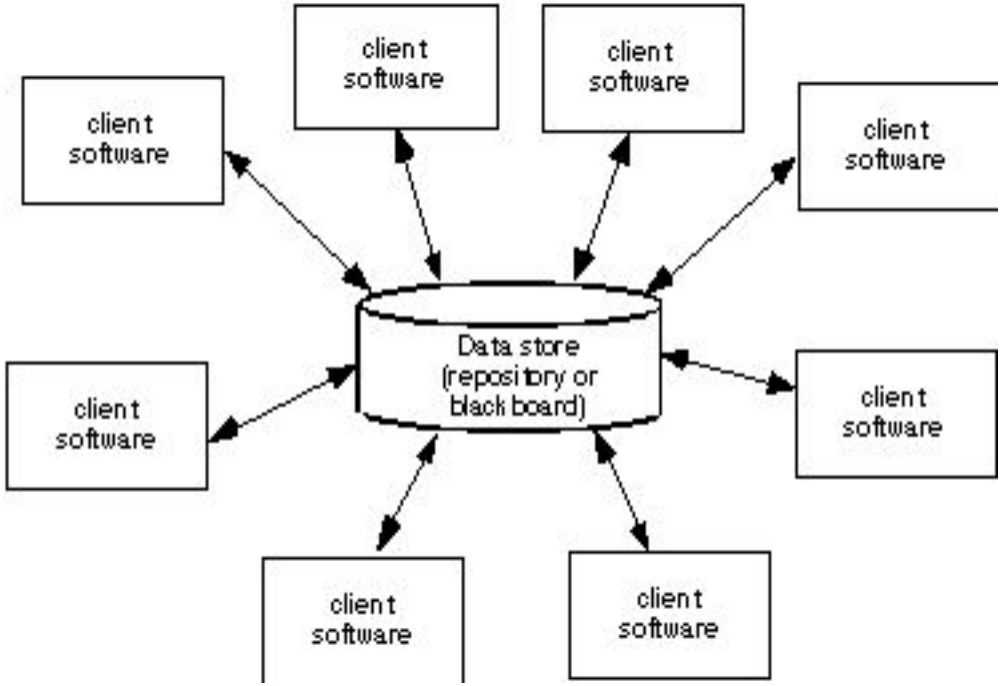
(2) a **set of connectors** that enable "communication, coordination and cooperation" among components,

(3) **constraints** that define how components can be integrated to form the system, and

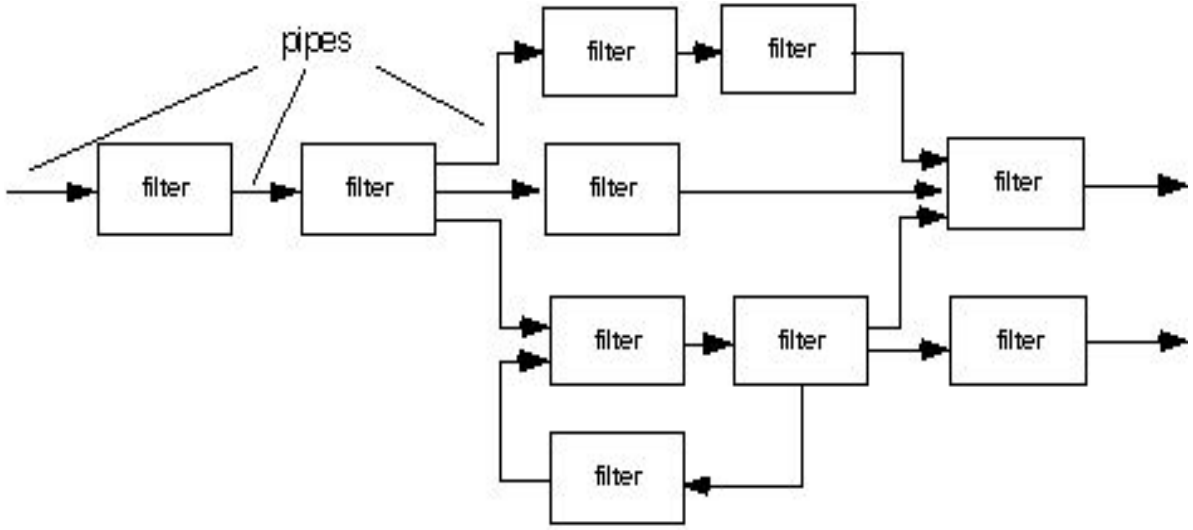
(4) **semantic models** that enable a designer to understand the overall properties of a system by analyzing the known properties of its constituent parts.

- Data-centered architectures
- Data flow architectures
- Call and return architectures
- Object-oriented architectures
- Layered architectures

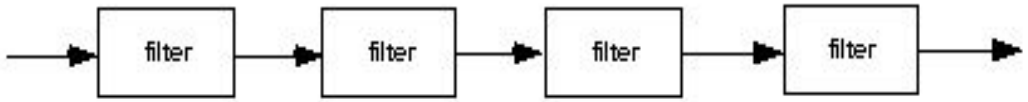
Data-Centered Architecture



Data Flow Architecture

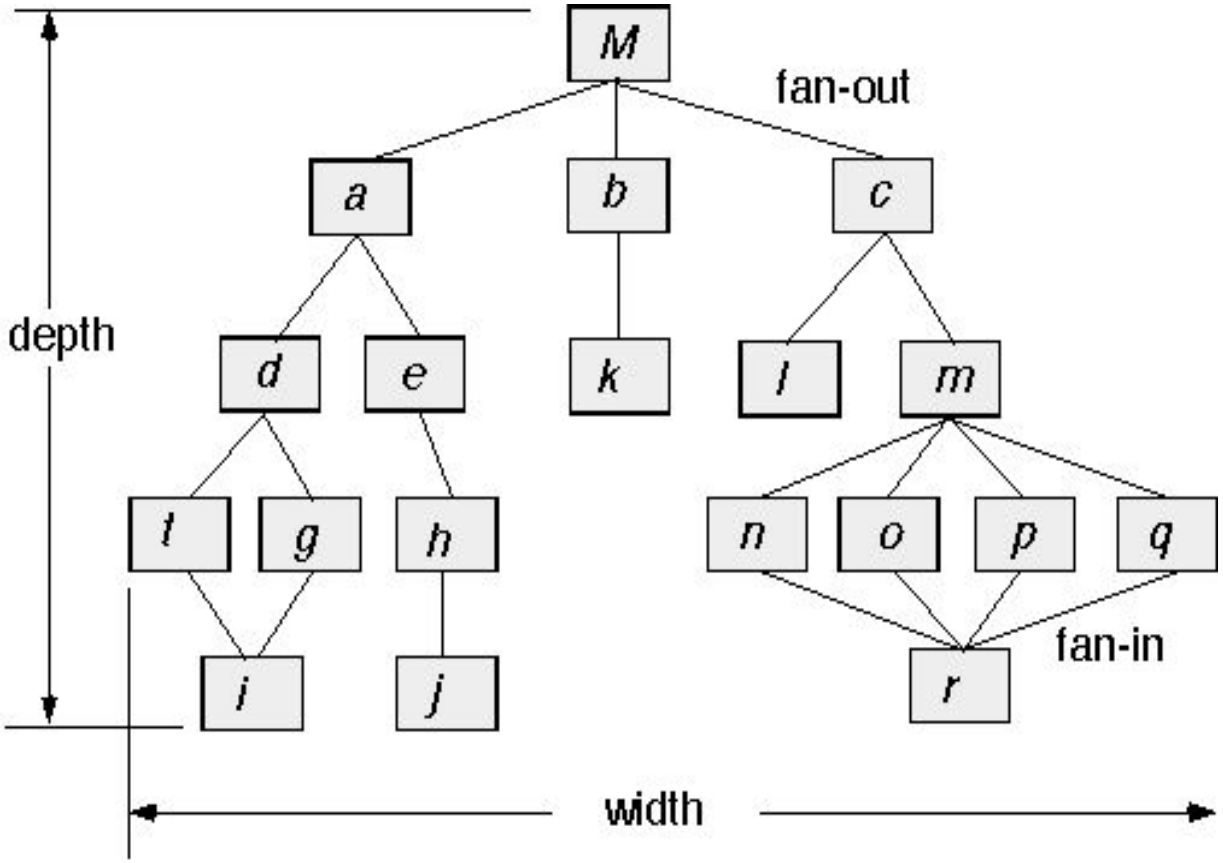


(a) pipes and filters

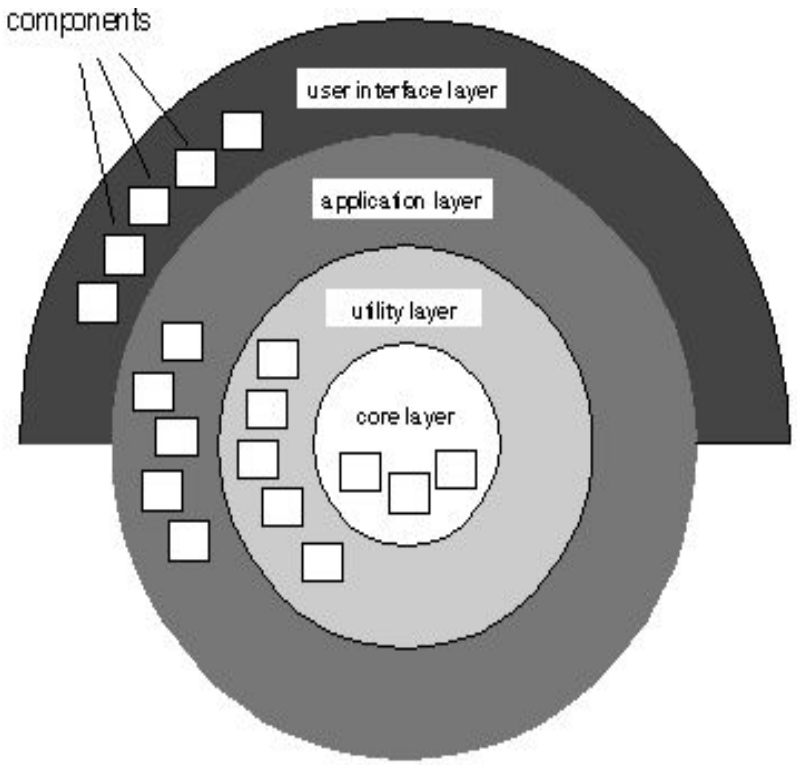


(b) batch sequential

Call and Return Architecture



Layered Architecture



Architectural patterns

- Architectural patterns address an application-specific problem within a specific context and under a set of limitations and constraints.
- The pattern proposes an architectural solution that can serve as the basis for architectural design.

Organization and Refinement

- It is important to establish a set of design criteria that can be used to assess an architectural design that is derived. The following questions [Bas03] provide insight into an architectural style:
  - **Control.** How is control managed within the architecture? Does a distinct control hierarchy exist, and if so, what is the role of components within this control hierarchy?
  - **Data.** How are data communicated between components? Is the flow of data continuous, or are data objects passed to the system sporadically? What is the mode of data transfer

Architectural Design

- Software to be developed must be put into context

  (i.e., define external entities and define the nature of interactions)
- Information acquired – In analysis model

  Information gathered – In requirement engineering
- Define and refine software components that implement architecture
- Continue process iteratively - until complete architectural structure has been derived.

Architectural Design

Representing the System in Context:

- System engineer must model the context
- Context diagram - models in which the system interacts with external entities
  (Uses ACD – Architectural Context diagram)
  (Input, Output, User interface, Processing)

- Systems that interoperate with the target system are represented as:
  - **Super ordinate systems** - using the target system as part of some higher level processing scheme
  - **Subordinate systems** - used by the target system to provide data or processing needed to complete the target system
  - **Peer level systems** - producing or consuming information needed by peers and the target system
  - **Actors** – those entities that interact with the target system by producing or consuming information that is necessary for requisite.
- Interfaces must be defined (relationship)
- All the data that flow into or out of the target system must be identified

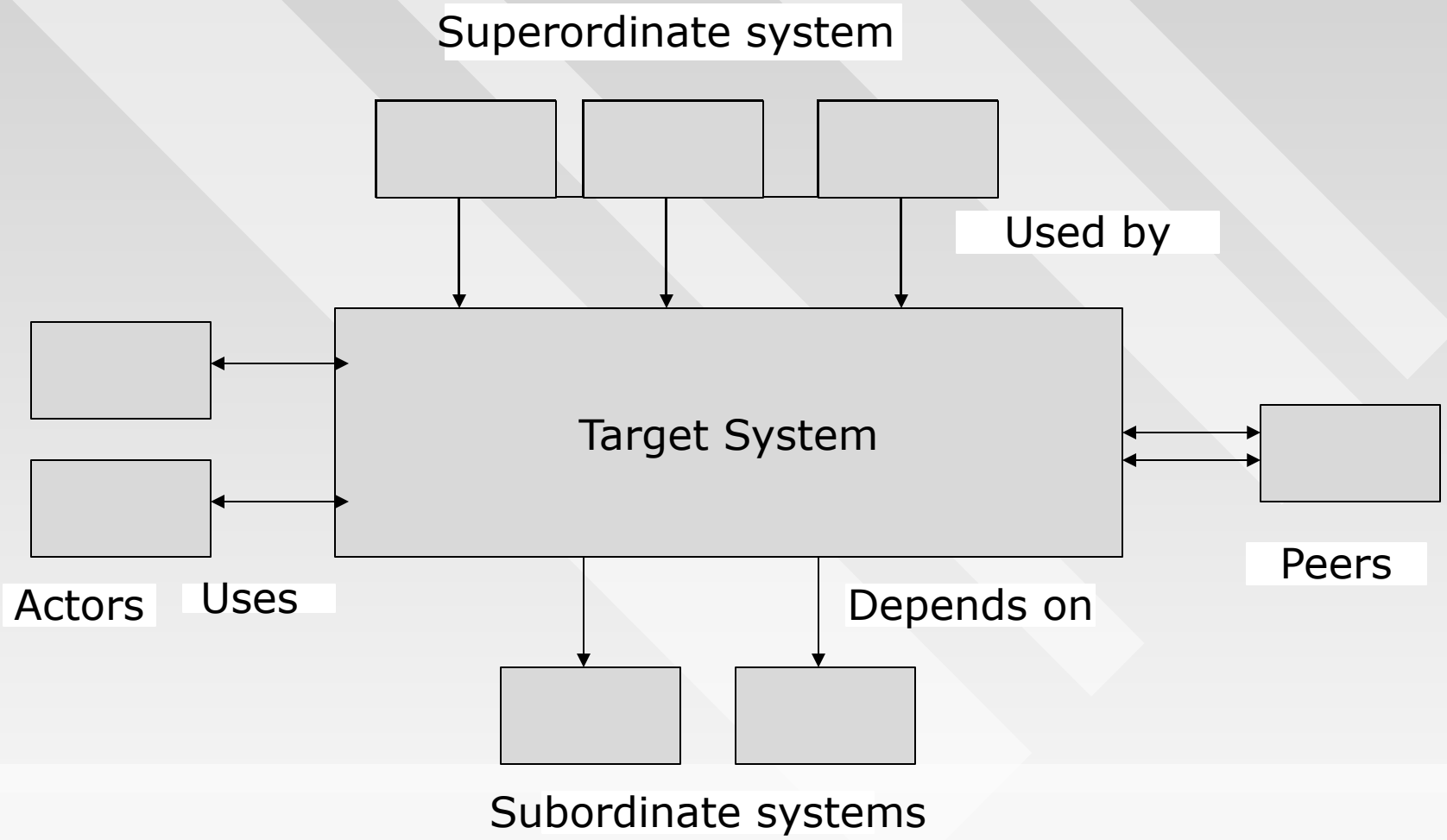


Fig: Architectural context diagram

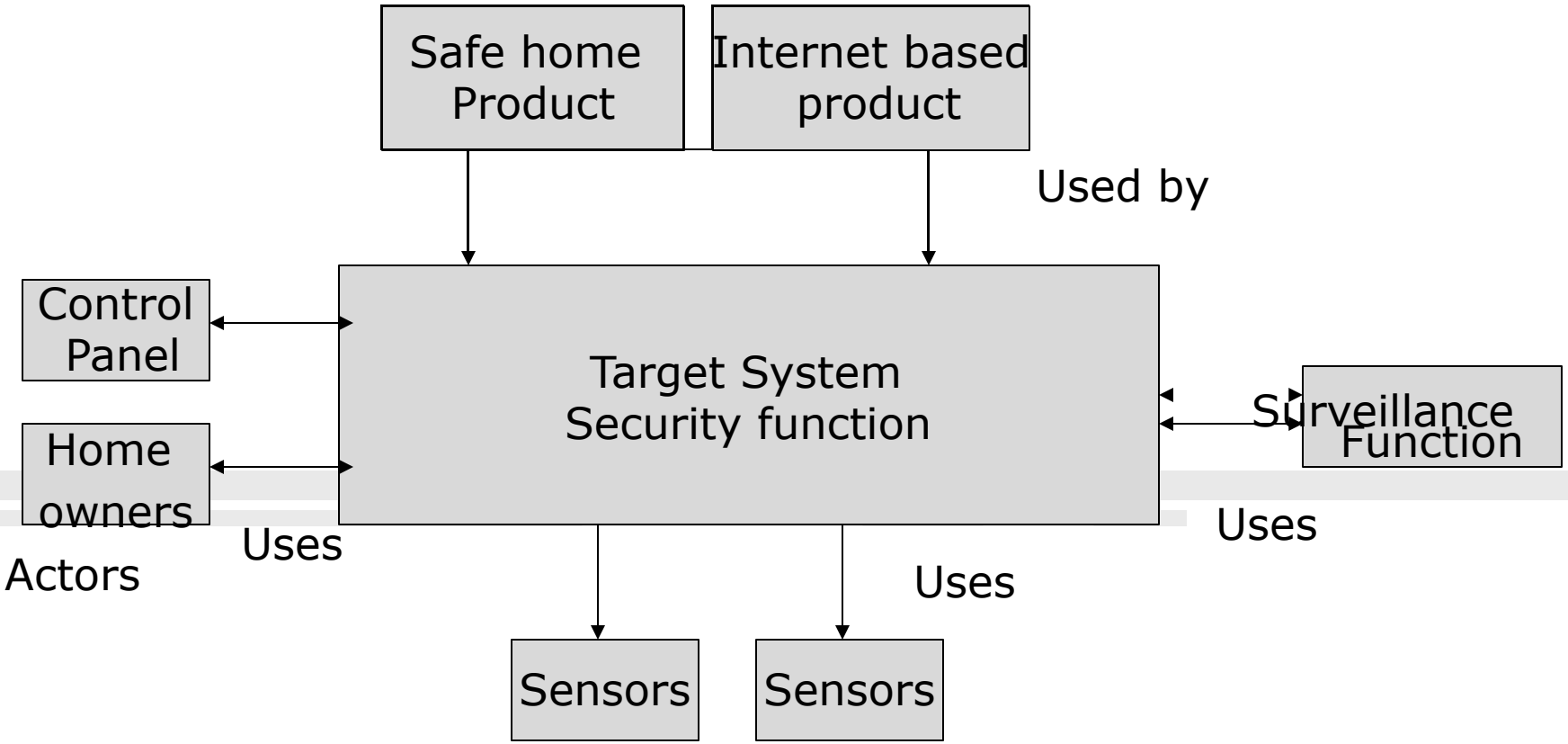


Fig: Architectural context diagram for the *SafeHome* security function

Defining Archetypes

- An archetype is a class or pattern that represents a core abstraction that is critical to the design of an architecture for the target system.
- The target system architecture is composed of these archetypes, which represent stable elements of the architecture but may be instantiated in many different ways.
- The following archetypes are
  - Node – represents a collection of input and output elements
  - Detector – an abstraction that encompasses all information into the target system
  - Indicator – an abstraction that represents all mechanisms for indication
  - Controller – an abstraction that depicts the mechanism that

ARCHETYPES

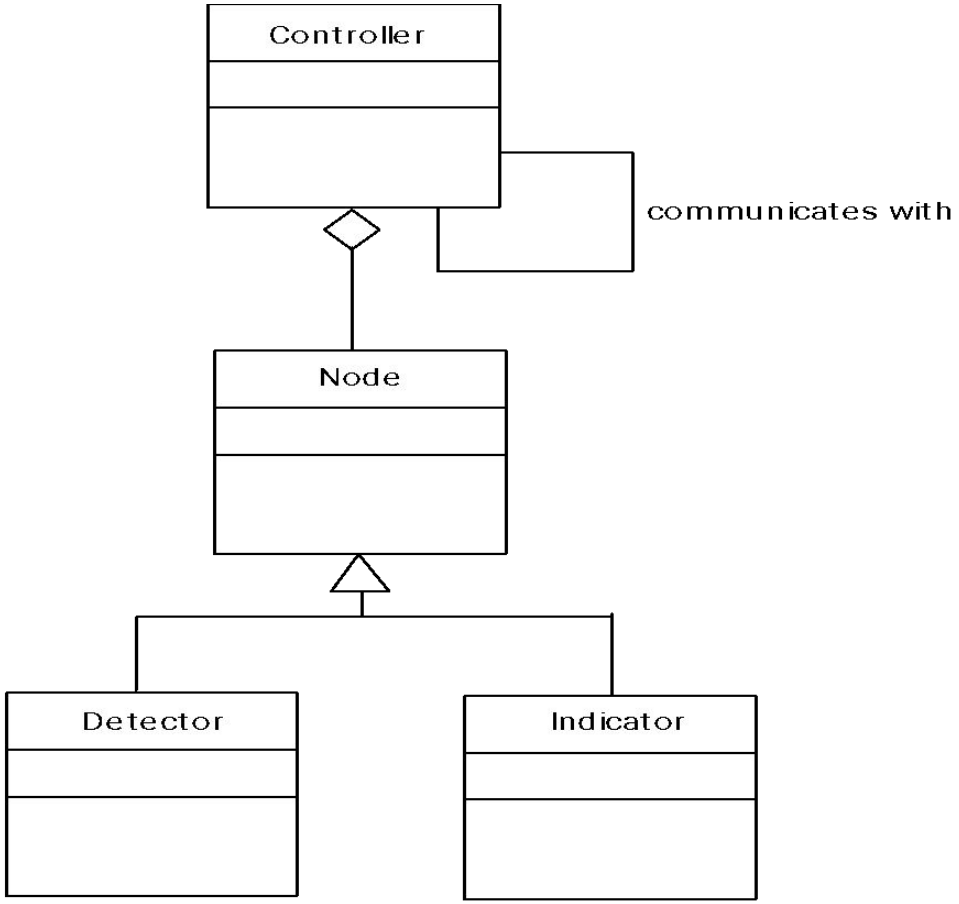
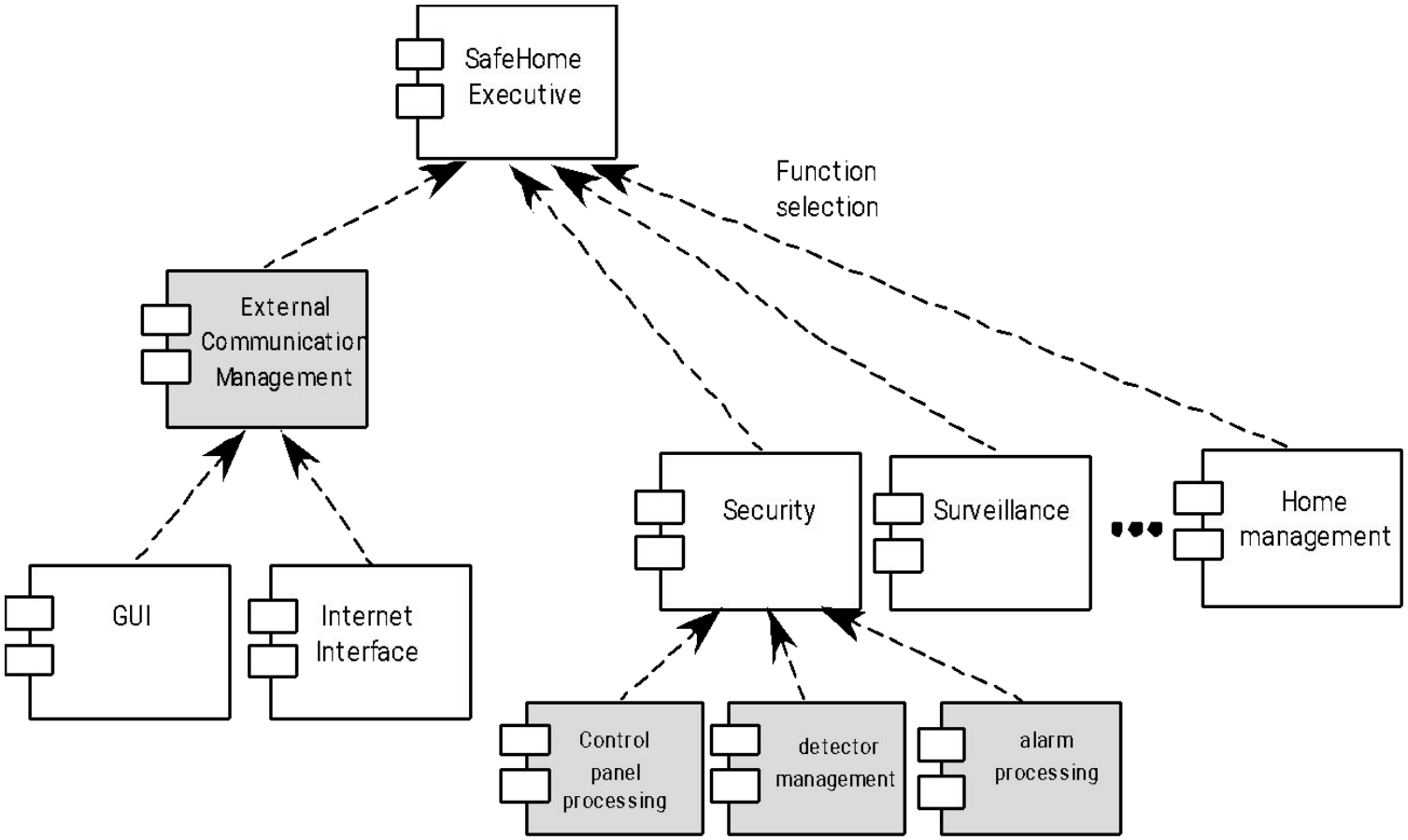
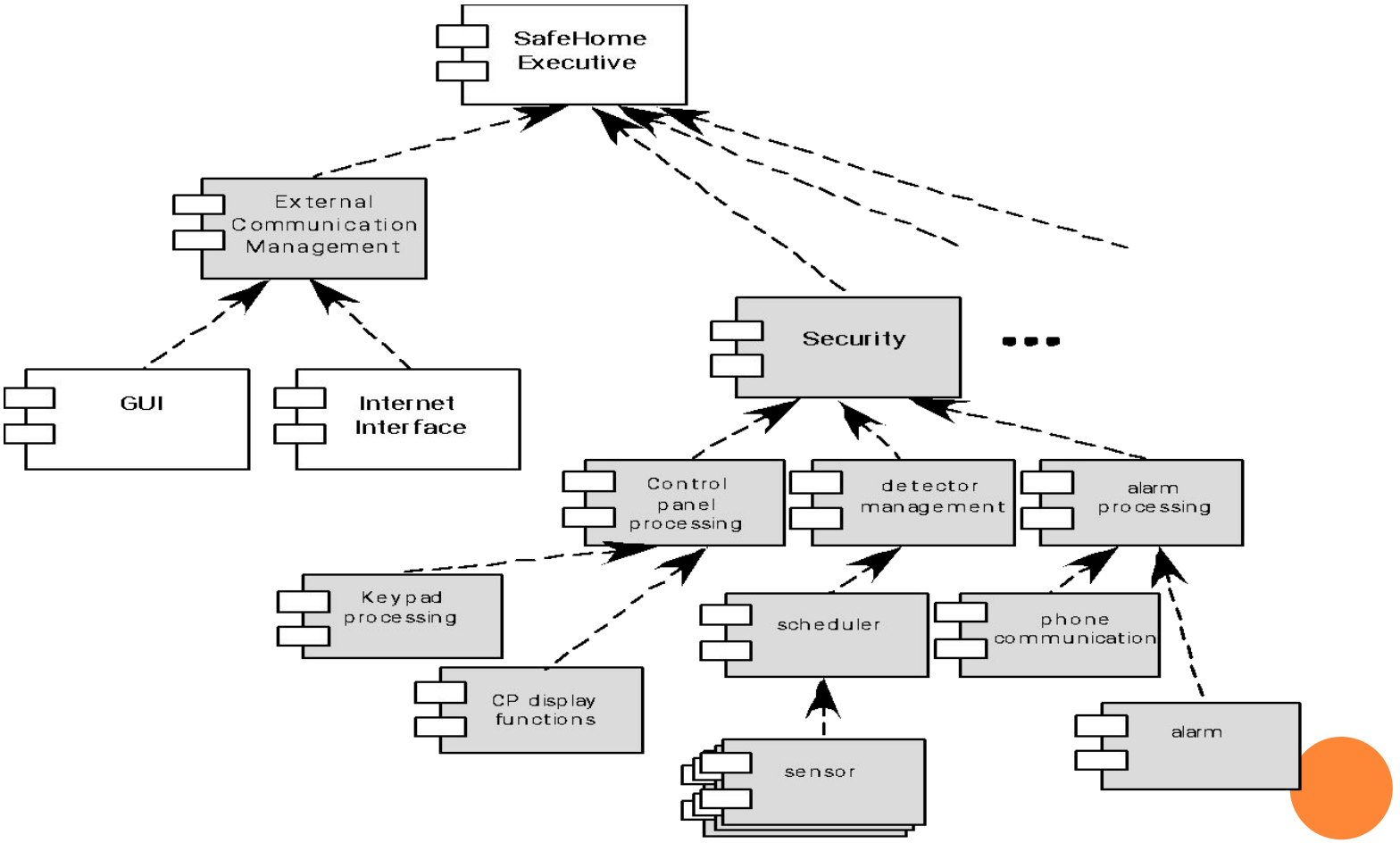


Figure 10.7 UML relationships for SafeHomesecurity function archetypes (adapted from [BOS00])

Component Structure



Refined Component Structure



Architectural Mapping using Data Flow

A mapping technique, called structured design, is often characterized as a data flow-oriented design method because it provides a convenient transition from a data flow diagram to software architecture.

Transition from DFD to program structure is as :

1. The type of information flow is established.
2. Flow boundaries are indicated.
3. DFD is mapped into program structure.
4. Control hierarchy is defined.
5. Resultant structure refined using design measures
6. Architectural description is refined and elaborated

The whole method is called information flow.

There are two types of flows:

1. Transform flow
2. Transaction flow

- **Transform flow** - data flow is sequential & flows in small number of straight line paths;
- **Transaction flow** - a single data item triggers information flow along one of many paths

Transform Flow:

- Information enters and exits the software in an "external world" form

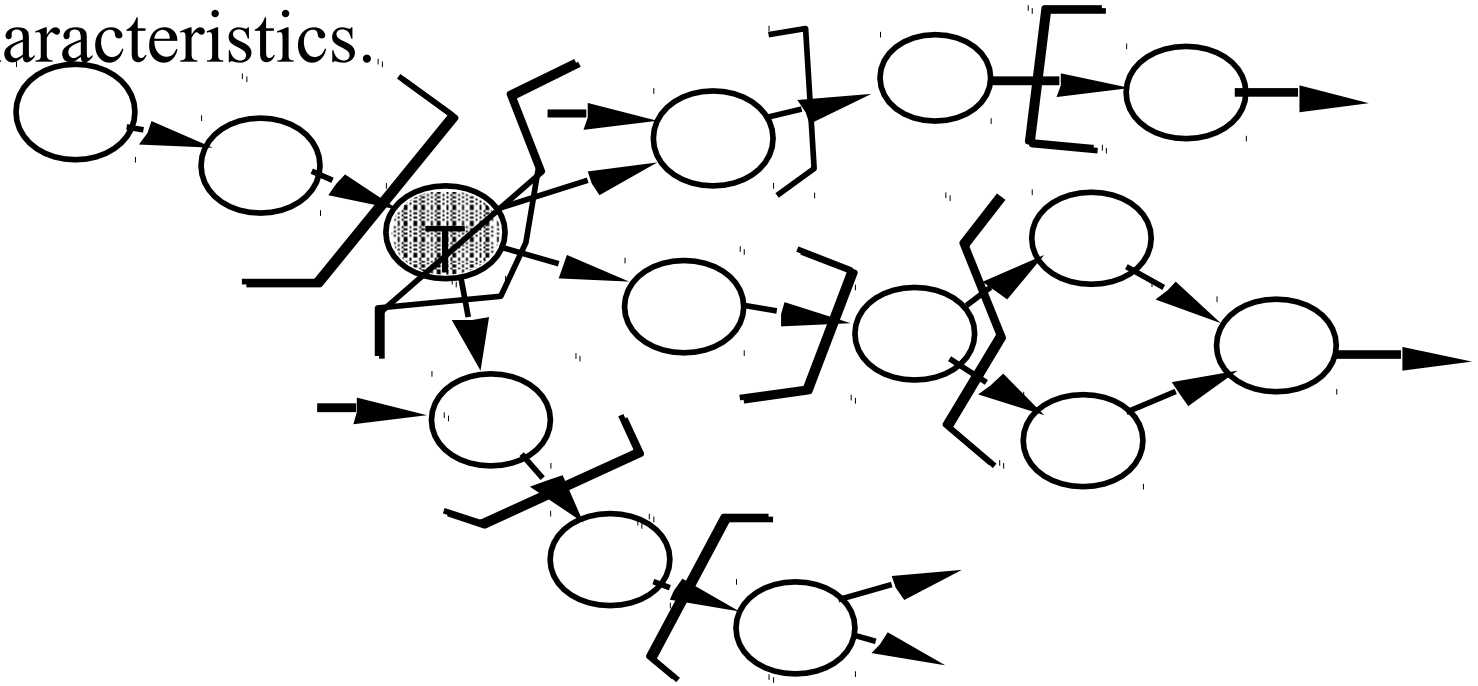
  (eg: data typed on a keyboard, voice commands to voice recognition systems, tones on a telephone line)
- This form of data cannot be understood by the software
- Externalized data converted into an internal form for processing.
- Data enters the software converted into internal form processed (transformed) and converted into external form to interact with the outside world.

- Two paths for conversion of data:
  - Incoming flow (path that does conversion of incoming data is called incoming flow)
  - Outgoing flow (path that converts data for sending out is called outgoing flow)
- Central part:
  - does actual processing (transformation) called transform center.
- Overall flow of data in a sequential manner
- Follows one, or only few, “straight line” paths.

Transaction Flow:

- Flows a single data item, called a transaction, but
- Triggers data flow along one of many paths i.e. there is branching.
- Data moving along an incoming path that converts external world information into a transaction.
- Transaction evaluated based on its value, flow along one of many action paths is initiated.
- Information flow - many action paths emanate called **transaction center**.
  - Eg. Receives a user command - control panel.

- DFD for large systems – transform + transaction
- Within a transaction flow we have one action path which would have transform flow.
- Eg: Process password action path in the user interaction subsystem shows transform flow characteristics.



Transaction Flow

Transform mapping

- A set of design steps allows DFD with transform flow characteristics to be mapped into a specific architectural style.

Design steps:

1. **Review the fundamental system model**
   - Data flow into and out of the function
   - i.e. consider the level 0 and level 1 DFD.

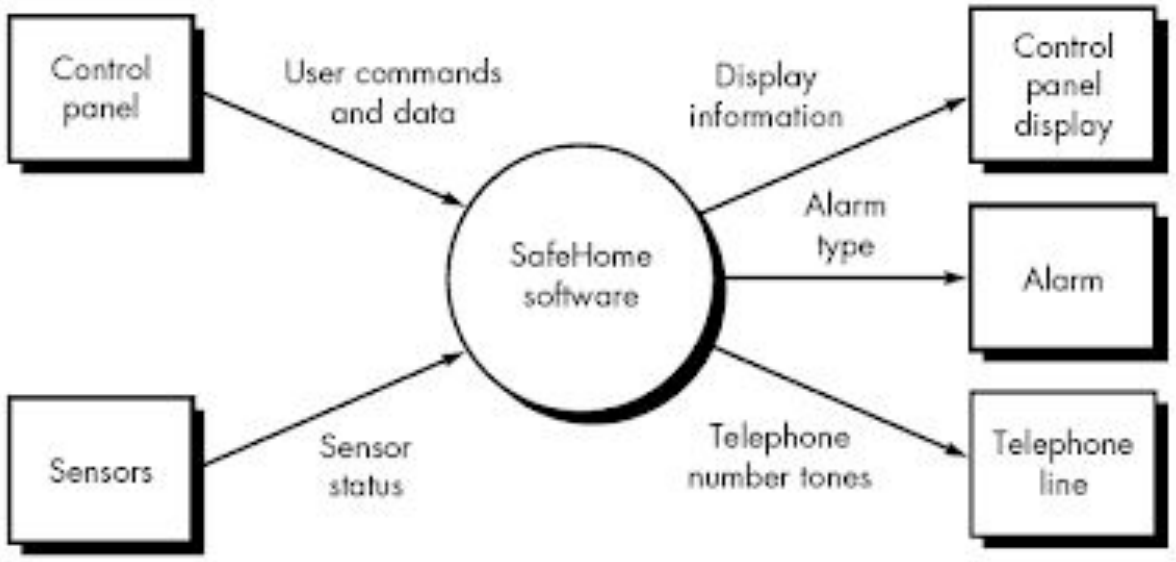


Fig: Context level DFD for the *SafeHome* security functon(Level 0 model)

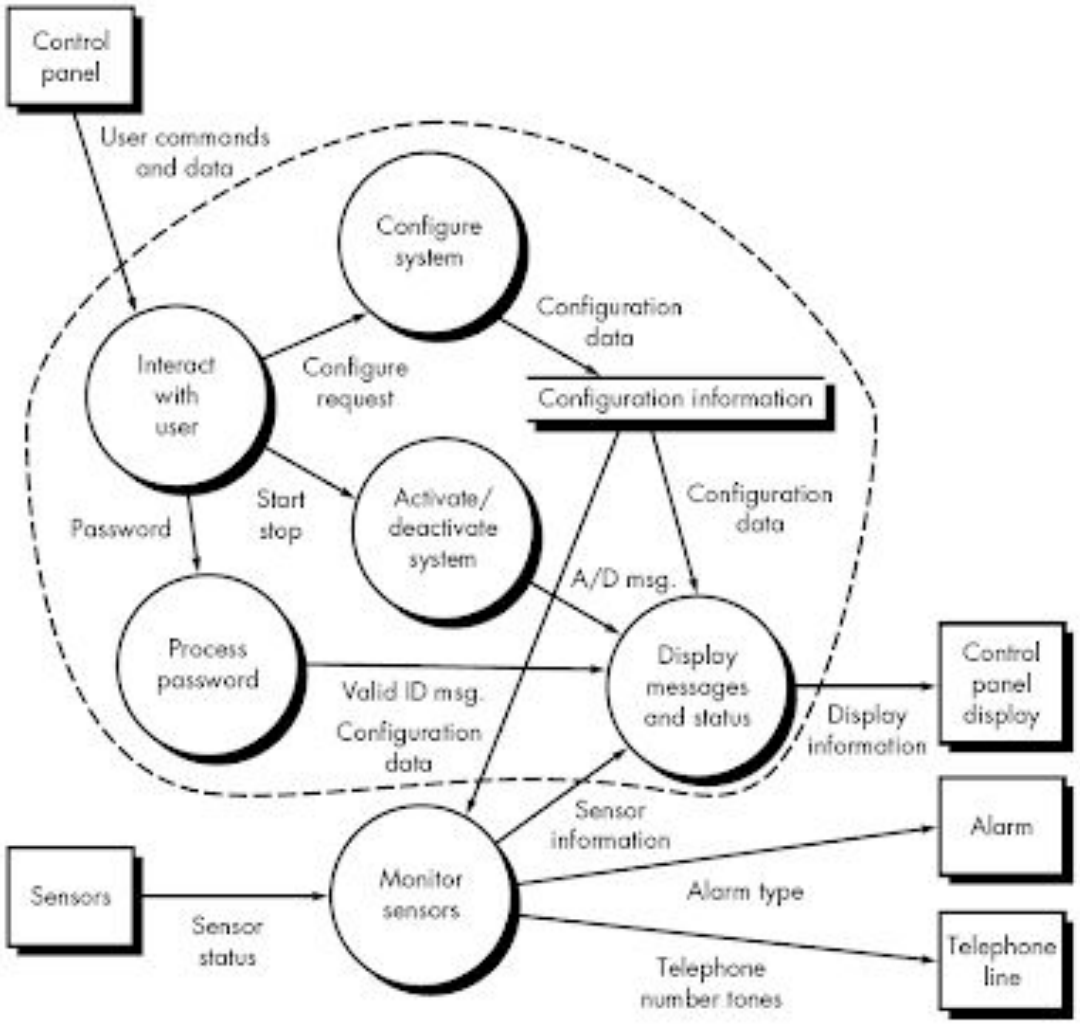
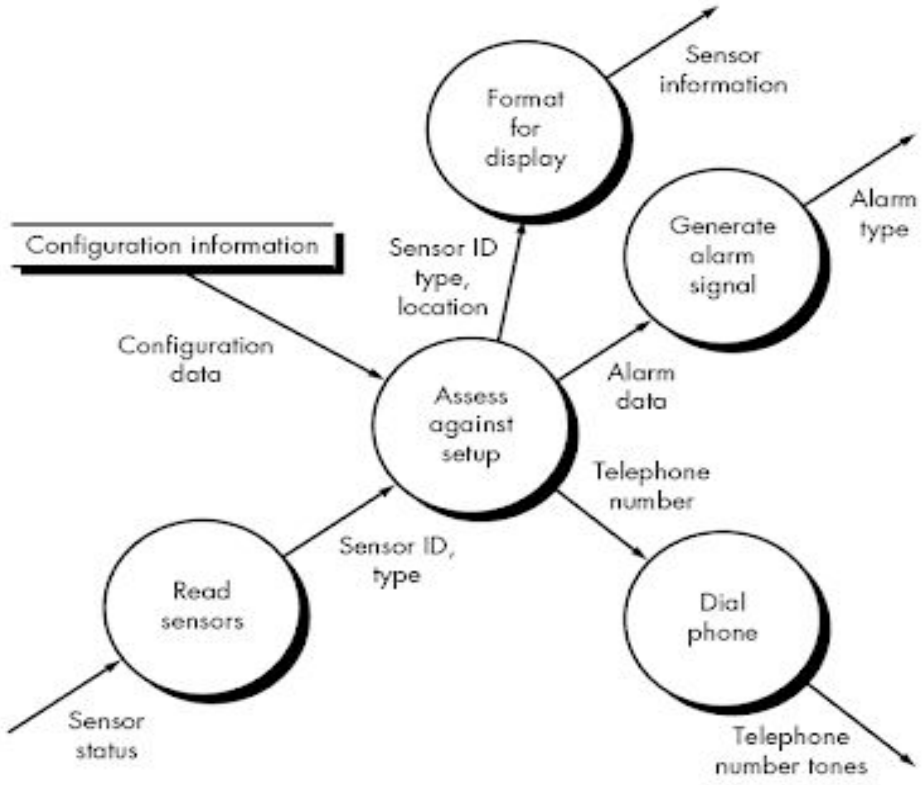


Fig: Level 1 DFD for the *SafeHome* security function

2. Review and refine DFD for the software:

- Information from analysis model refined
- i.e. refine DFD to get level 2 and 3 DFD.
- At level 3 of DFD:
- each transform exhibits high level of cohesion

-process implied by a transform performs a single function that can be implemented as a component.



3. Determine whether the DFD has transform or transaction flow characteristics:

- Incoming data: only one

- outgoing : appears to be three

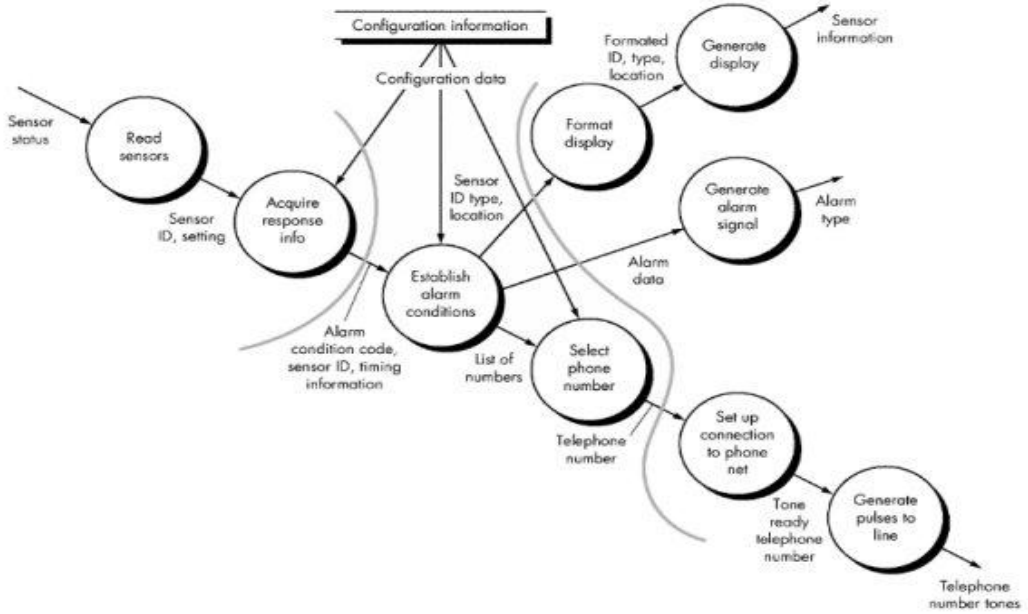
  alarm condition

  second one is for display (Sensor, telephone)

  so action path is only one.

  Hence it can be considered as transform flow.

4.Isolate the transform center by specifying incoming and outgoing boundaries:



5. **Perform "first level factoring":**

- We know that there is transform flow
- We divide structure in 3 different controls

(i) incoming path

(ii) transform

(iii) outgoing path.

-One main controller called monitor sensors executive which controls and supervises three controllers

(sensor input controller, alarm conditions controller and alarm output controller)

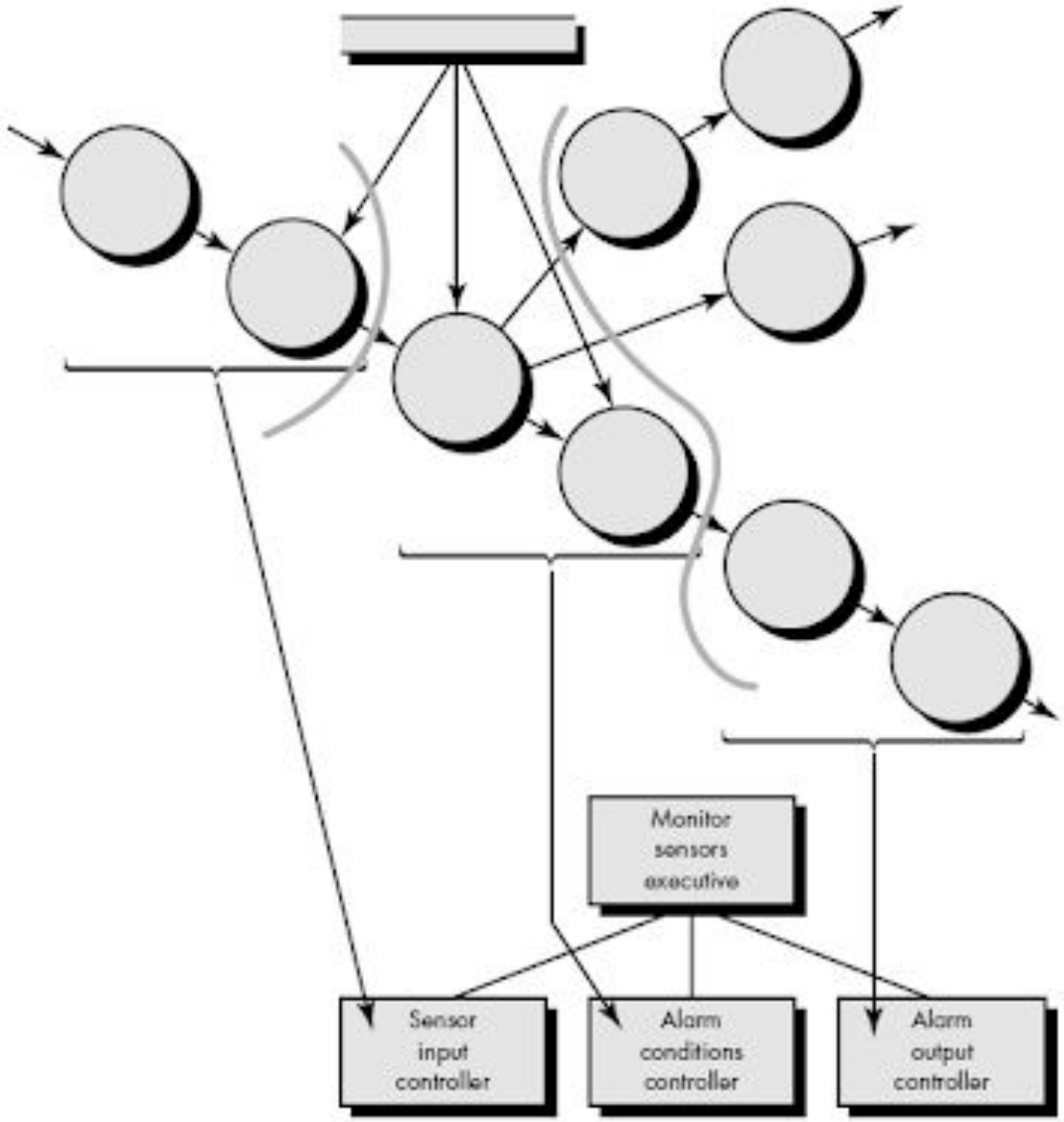


Fig: First-level factoring for monitor sensors

6. **Perform second level factoring:**

- Assign one program component to each bubble

-Decide the modules which are going to do actual work.

-Using controls decided in the first level of factoring

-Worker modules decided in the second level of factoring

- Decide the complete program structure.

- This is the first iteration architecture.

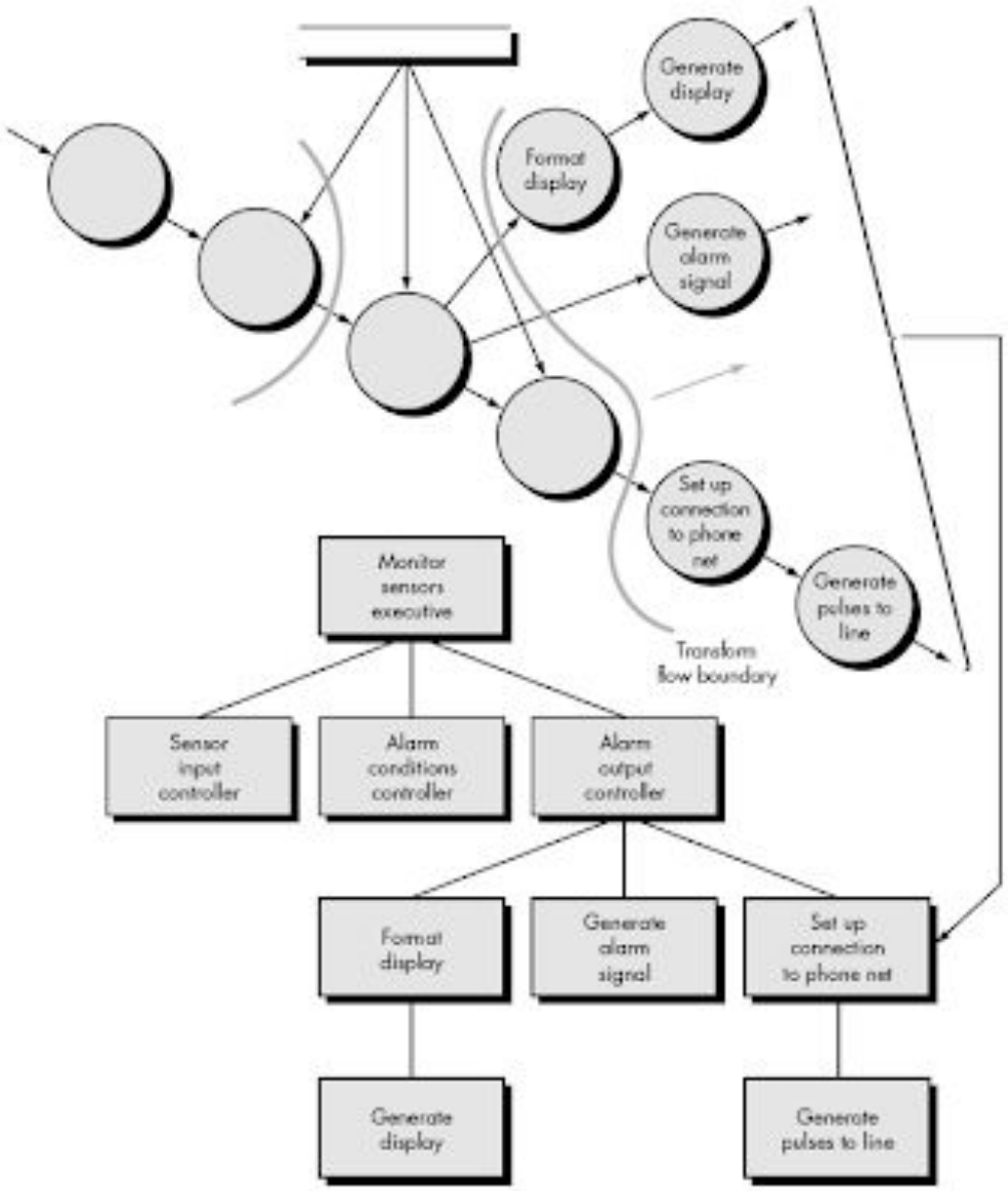


Fig: Second-level factoring for monitor sensors

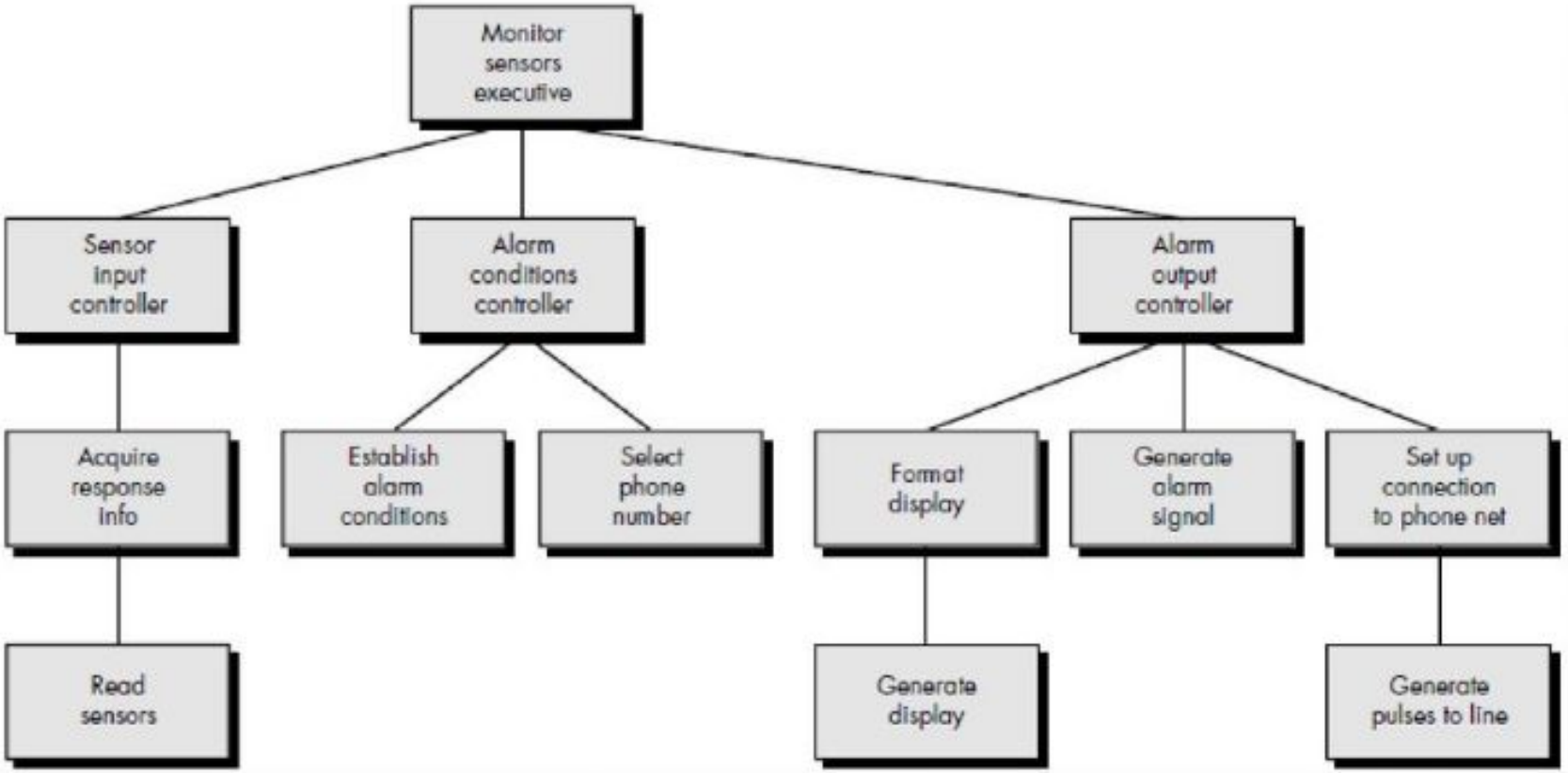


Fig: First iteration structure for monitor sensor

7. **Refine the first iteration architecture using design heuristics for improved software quality:**

- **R**efinement modules are to produce
  - sensible factoring
  - good cohesion
  - minimal coupling
  - a structure that can be implemented without difficulty
  - tested without confusion
  - maintained
- Sensor input controller and Alarm conditions controller areremoved and Format display and generate display are combined to give Produce display.

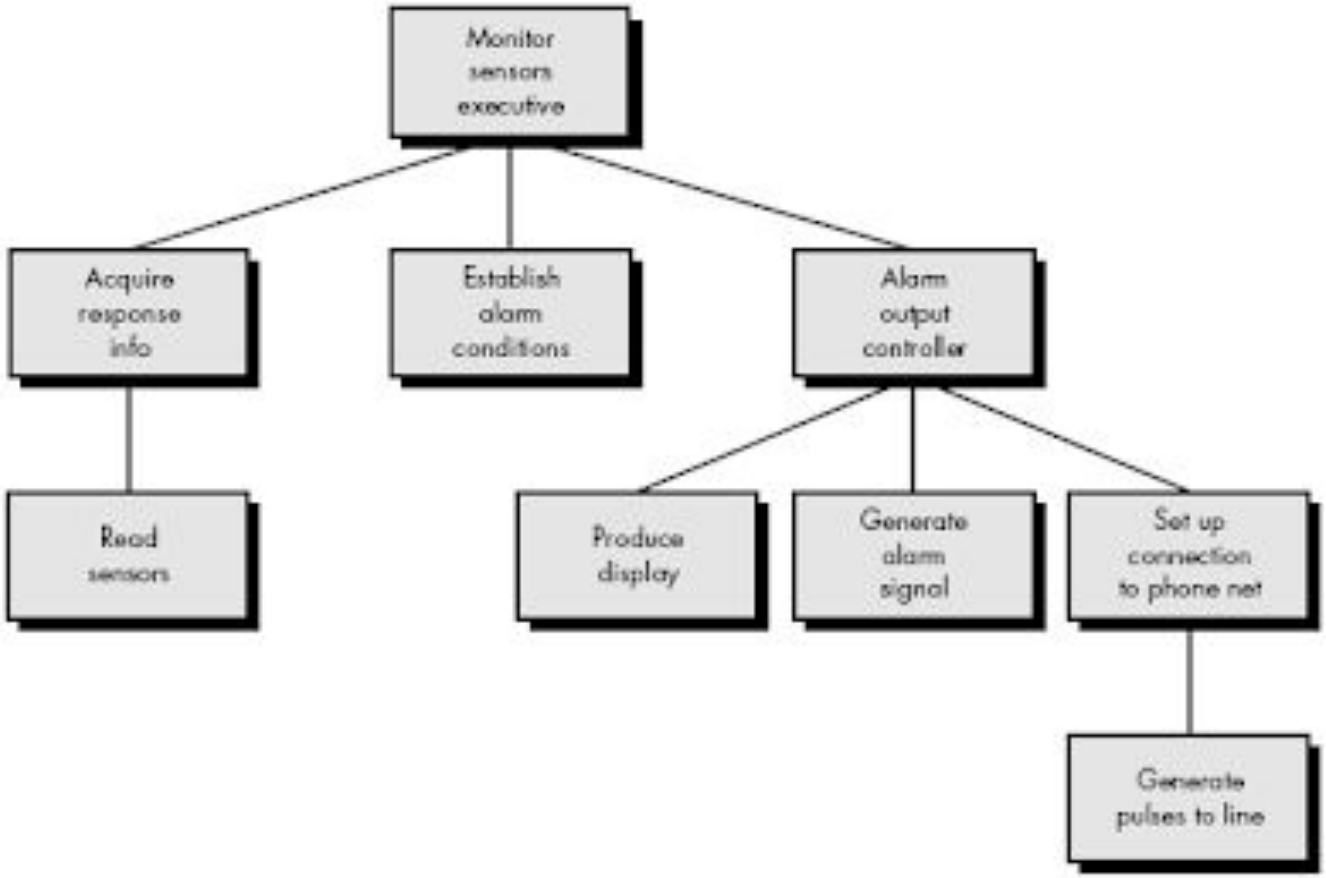


Fig: Refined program structure for monitor sensors

Refining the Architectural Design

- Any discussion of design refinement should be prefaced with the following comment:

  "Remember that an 'optimal design' that doesn't work has questionable merit."
- Refinement of software architecture during early stages of design is to be encouraged.
- Alternative architectural styles may be derived, refined, and evaluated for the "best" approach.
- This approach to optimization is one of the true benefits derived by developing a representation of software architecture.
- Structural simplicity often reflects both elegance and efficiency.
- Design refinement should strive for the smallest number of components that is consistent with effective modularity and the least complex data structure that adequately serves information requirements