Software Engineering: A Practitioner's Approach, 6/e

Bab 10
Desain Arsitektur

R.S. Pressman & Associates, Inc.
A. Haidar Mirza, M.Kom
Siti Sa'uda, M.Kom.

Kenapa Arsitektur ?

Arsitektur bukanlah PL operasional, namun dia merupakan representasi yang memungkinkan pengembang PL untuk : (1)menganalisa efektivitas desain dalam memenuhi kebutuhan,

(2) Mengetahui alternatif2x arsitektur pada keadaan dimana membuat perubahan desain masih relatif lebih mudah, dan

(3) Mengurangi resiko terkait dengan konstruksi PL.

Mengapa Arsitektur Penting?

- Representasi dari arsitektur PL adalah enabler bagi komunikasi antar pihak (stakeholder) yang tertarik dengan pengembangan sistem berbasis komputer.
- Arsitketur menyoroti keputusan desain awal yang akan mempunyai pengaruh yang sangat besar pada pekerjaan RPL yang mengikutinya, dan keberhasilan pada entitas sistem operasional.
- Arsitektur membangun model yang relatif kecil dan mudah digenggam secara intelektual tentang bagaimana sistem distrukturkan dan bagaimana komponen2x bekerja sama [BAS03].

Desain Data

- Pada level arsitektur …
  - Desain satu atau lebih database untuk mendukung arsitektur aplikasi
  - Desain method untuk ‘mining’ isi dari berbagai database
    - Navigasi melalui database2x yang ada dalam usaha untuk mengambil informasi level bisnis yang sesuai
    - Desain sebuah data warehouse—sebuah database besar, independen yang mempunyai akses pada data yang disipan dalam database yang melayani sekelompok aplikasi yang dibutuhkan bisnis

Desain Data

- Pada level komponen …
  - Mengambil objek2x data dan mengembangkan satu set abstraksi data
  - Implementasi atribut2x objek data sebagai satu atau lebih struktur data
  - review struktur data untuk memastikan bahwa relasi yang tepat sudah dibuat
  - Sederhanakan struktur data sesuai dengan kebutuhan

Desain Data—Level Komponen

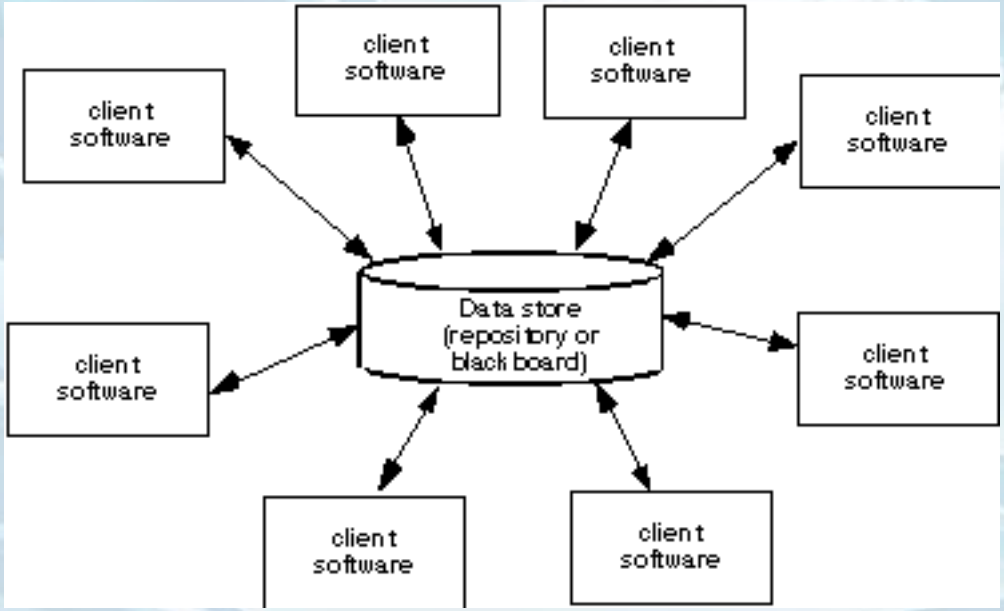
1. Prinsip2x analisis semantik yang diterapkan pada fungsi dan perilaku harus juga dapat berjalan pada data.
2. Seluruh struktur data dan operasi yang akan dilakukan harus dapat diidentifikasi.
3. Sebuah data dictionary harus dibuat dan digunakan untuk menentukan desain program dan data.
4. Keputusan desain data level rendah harus ditunda hingga akhir proses desain.
5. Representasi struktur dara harus diketahui oleh modul yang menggunakannya langsung dalam struktur tersebut (enkapsulasi).
6. Sebuah pustaka struktur data dan operasi yang memungkinkan untuk diterapkan harus dikembangkan.
7. Desain PL dan bahasa pemrograman harus mendukung spesifikasi dan realisasi dari tipe data abstrak.

Ragam Gaya Arsitektur

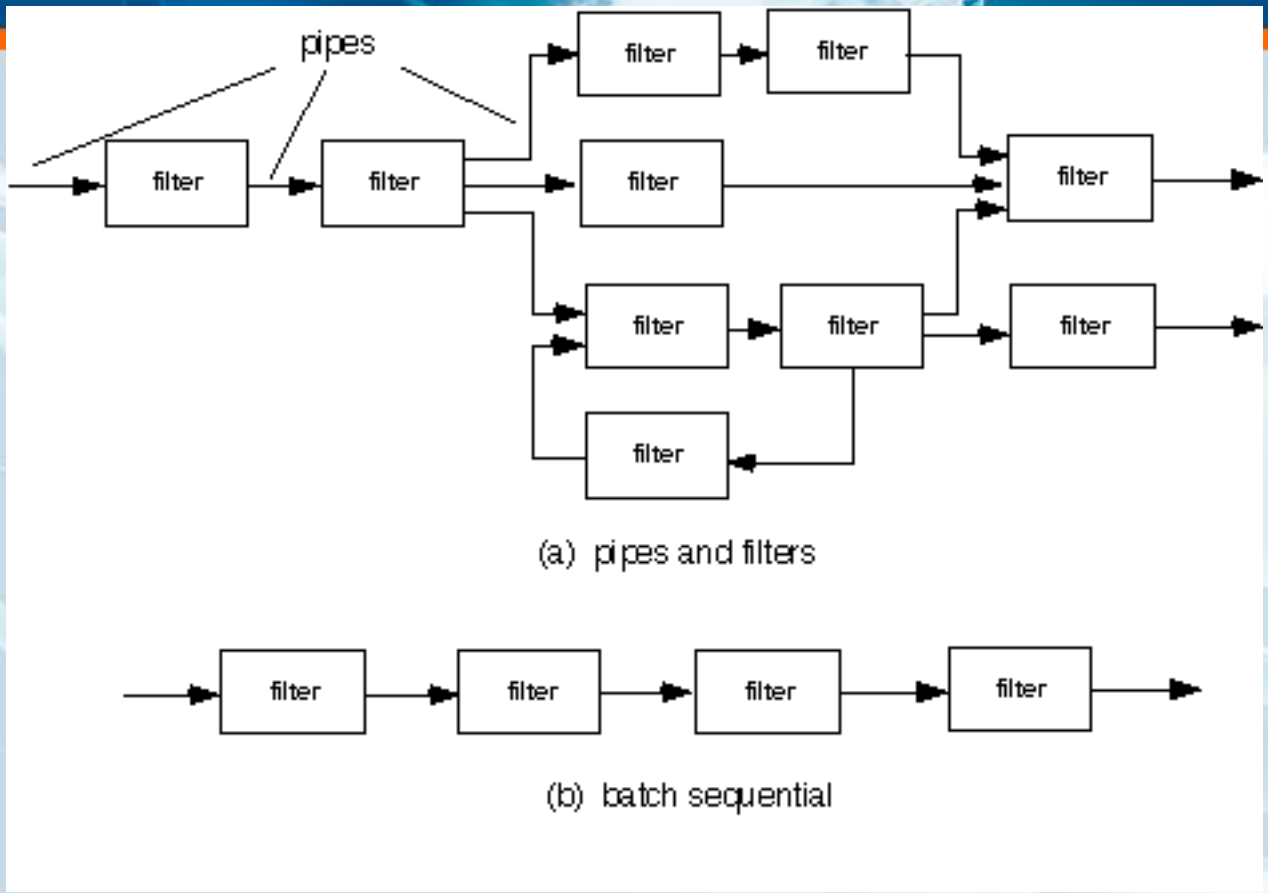
Masing2x menggambarkan kategori sistem yang menunjukkan : (1) a sekumpulan komponen (mis database, modul komputasi) yang menunjukkan fungsi yan dibutuhkan sistem, (2) sekumpulan connectors yang memungkinkan komunikasi, koordinasi dan kerjasama antar komponen components, (3) batasan yang menentukan bagaimana komponen dapat diintegrasikan untuk membentuk sistem, dan (4) smodel semantik yang memungkinkan desainer untk memahami properti keseluruhan dari sistem dengan menganlisai properti dalam bagian2x di dalamnya.

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

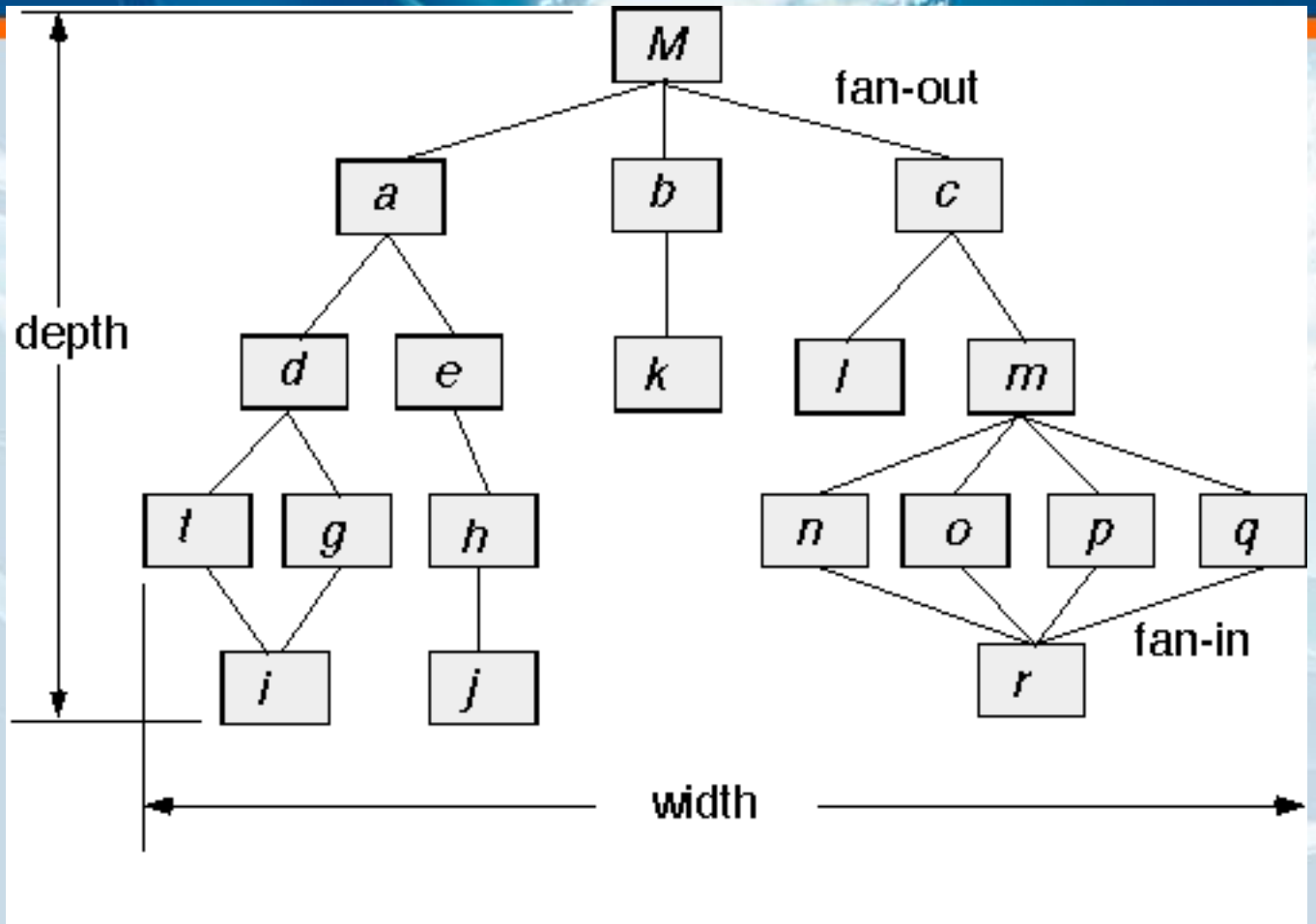
Data-Centered Architecture



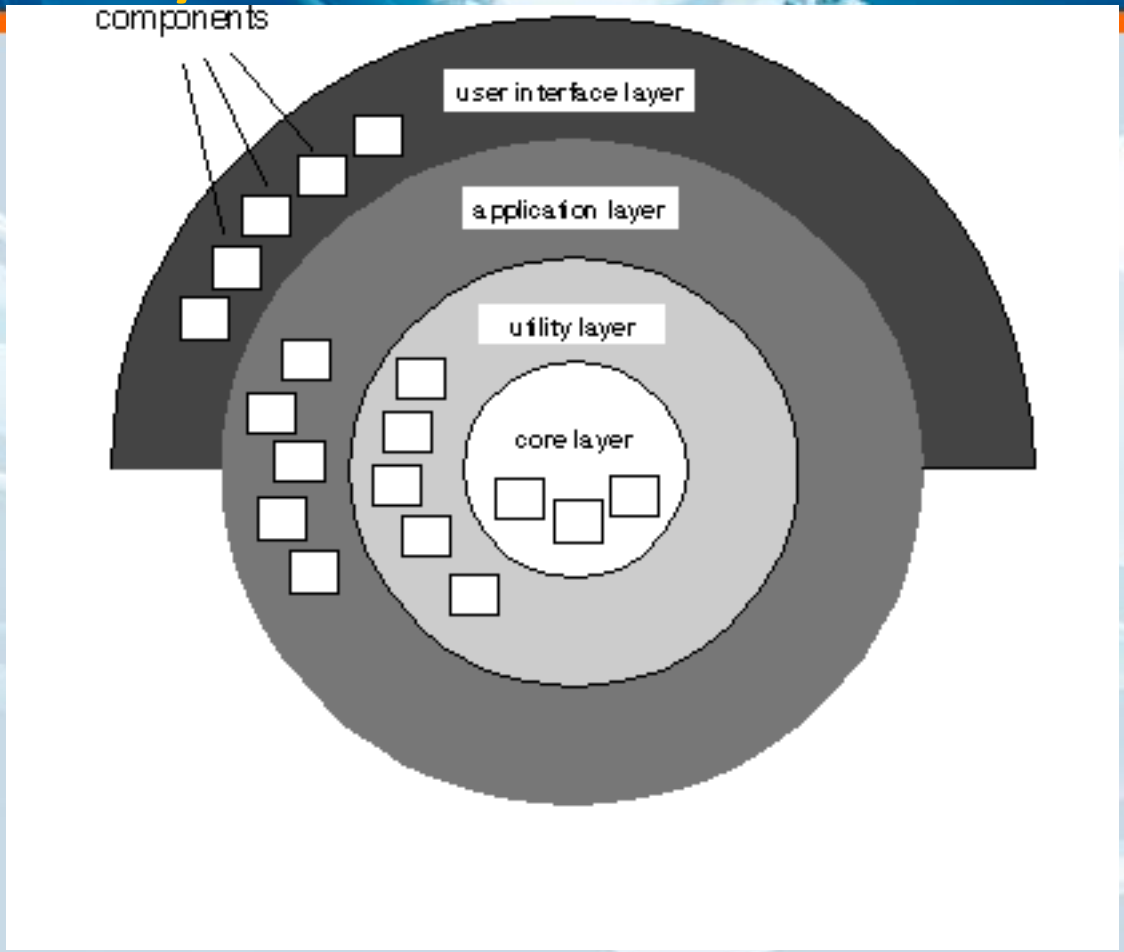
Data Flow Architecture



Call and Return Architecture



Layered Architecture



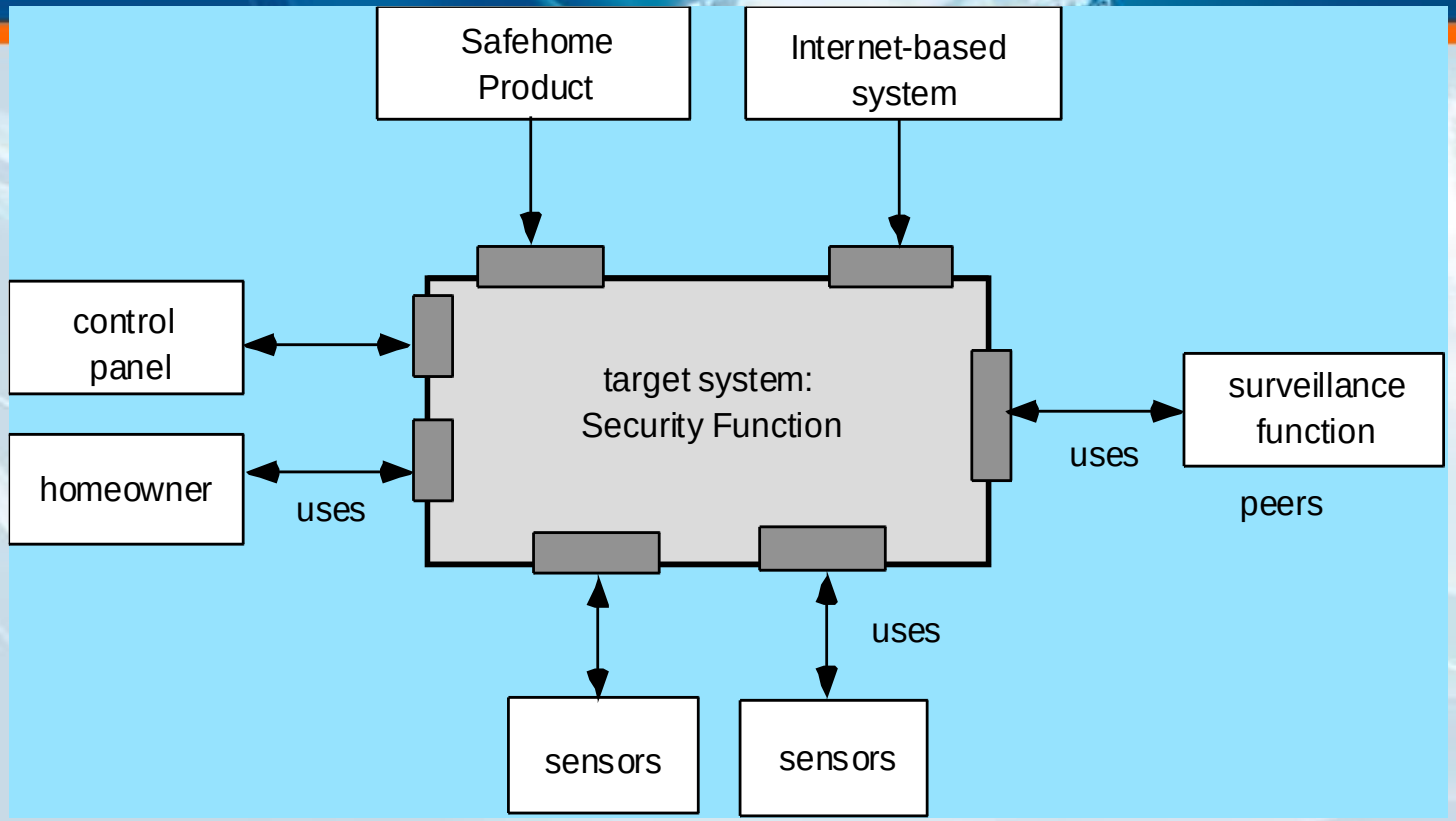
Pattern Arsitektural

- Concurrency—aplikasi harus menangani banyak tugas dalam pola yang mensimulasikan paralelisasi
  - *operating system process management* pattern
  - *task scheduler* pattern
- Persistence—Data ada jika dia bertahan setelah eksekusi proses yang membuatnya. Ada dua pattern umum ::
  - *database management system* pattern yang menerapkan penyimpanan dan pengambilan dari DBMS kepada arsitektur aplikasi
  - *application level persistence* pattern yang membangun fitur persistence pada aristektur aplikasi
- Distribution— pola dimana sistem atau komponen2x di antaranya berkomunkasi dalam lingkungan terdistribusi
  - *broker* bertindak sebagai orang di tengah antara komponen klient dan komponen server.

Desain Arsitektur

- PL harus ditempatkan pada konteks
  - Desain harus menentukan entitas eksternal (sistem lain, piranti, orang) dimana PL berinteraksi dengannya
- Sekumpulan arsitektur archetypes harus diidentifikasi
  - *archetype* adalah abstraksi (mirip dengan class) yang menampilkan satu elemen dari perilaku sistem
- Desainer menentukan struktur sistem dengan memilih komponen PL yang mengimplmentasi masing2x archetype

Architectural Context



Archetypes

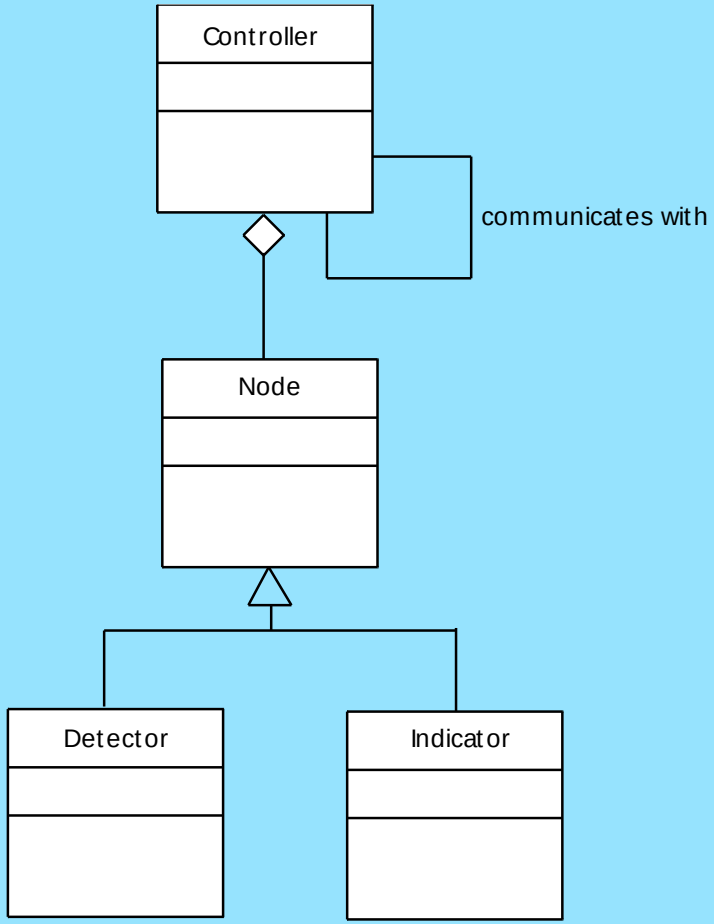
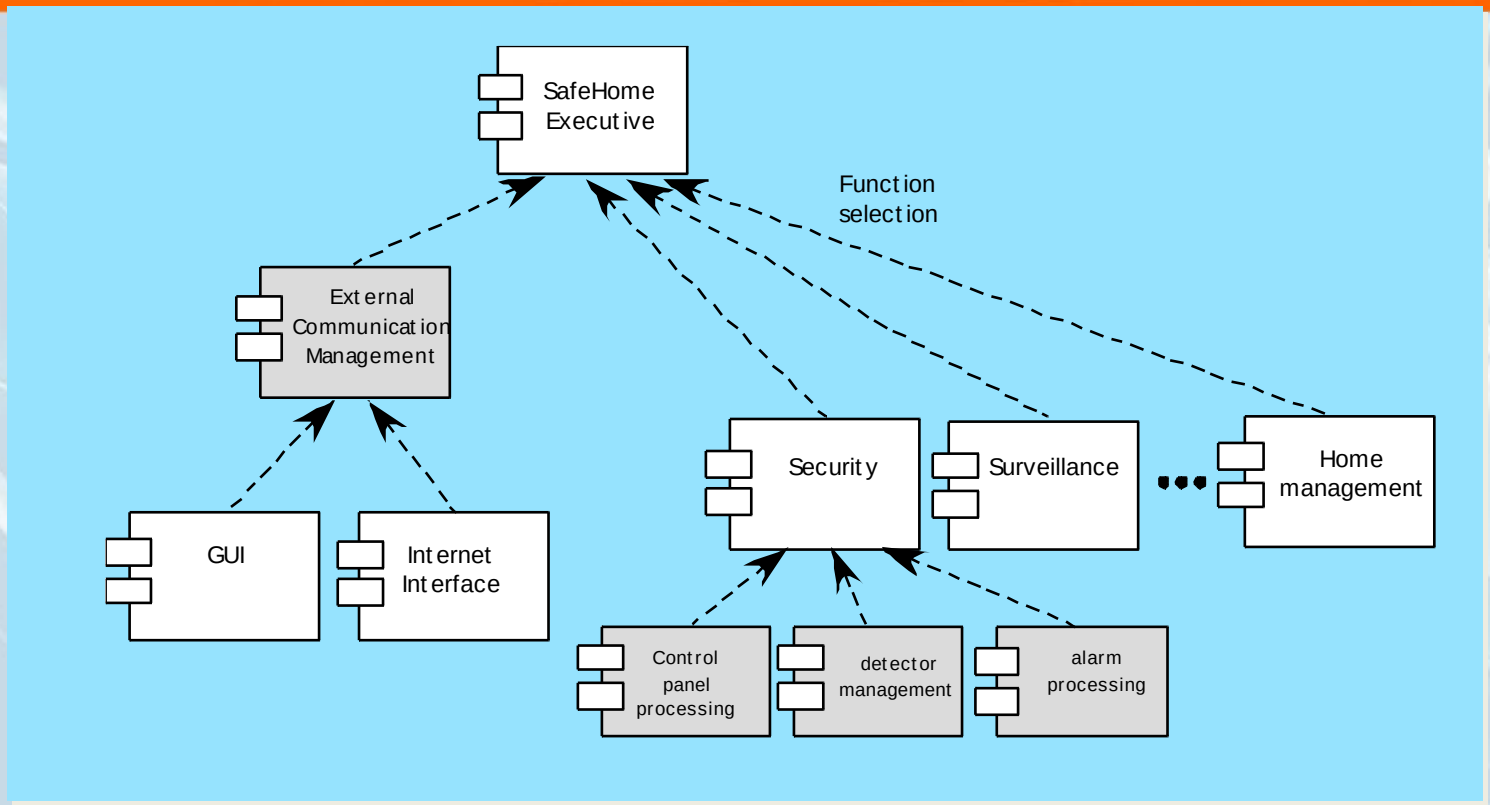
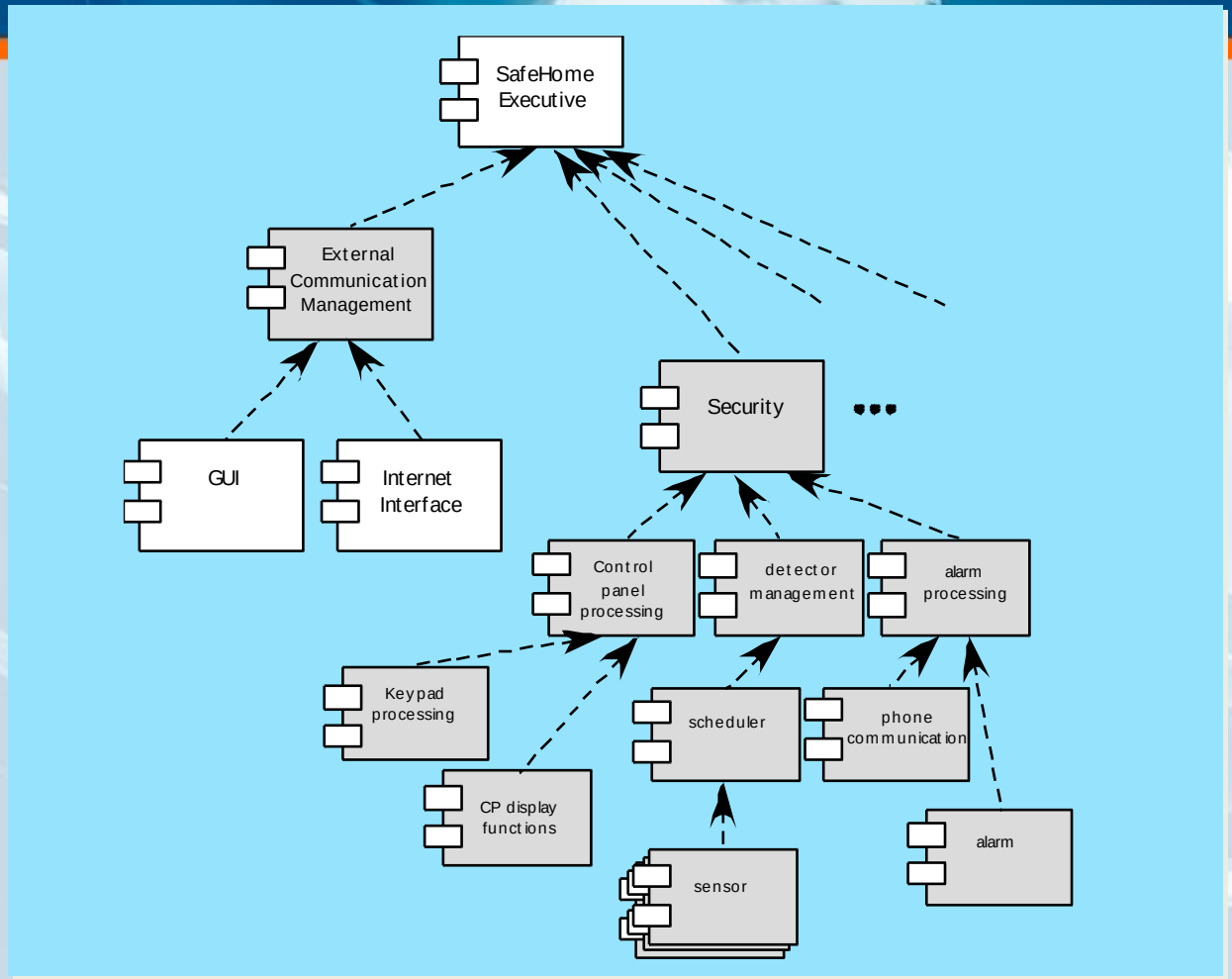


Figure 10.7 UML relationships for SafeHome security function archetypes (adapted from [BOS00])

Component Structure



Refined Component Structure

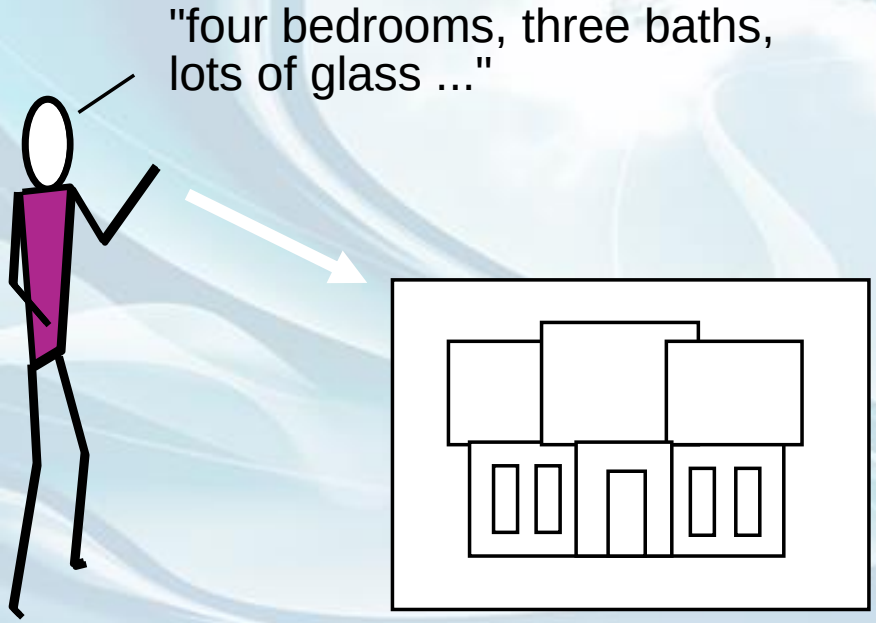


Analisis Desain Arsitektur

1. Kumpulkan semua skenario.
2. Dapatkan kebutuhan2x, batasan2x, dan gambaran lingkungan.
3. Gambarkan pola/gaya arsitektur yang telah dipilih untuk menangani skenario2x dan kebutuhan2x ::
   - module view
   - process view
   - data flow view
4. Evaluasi kualitas atribut2x dengan melihat setiap atribut dalam isolasi.
5. Kenali kualitas atribut untuk setiap atribut arsitektural untuk masing-masik gaya arsitektur yang spesifik.
6. Lakukkan kritik pada arsitektur2x kandidat (yg dikembangkan pada langkah 3) menggunakan analisis pada langkah 5.

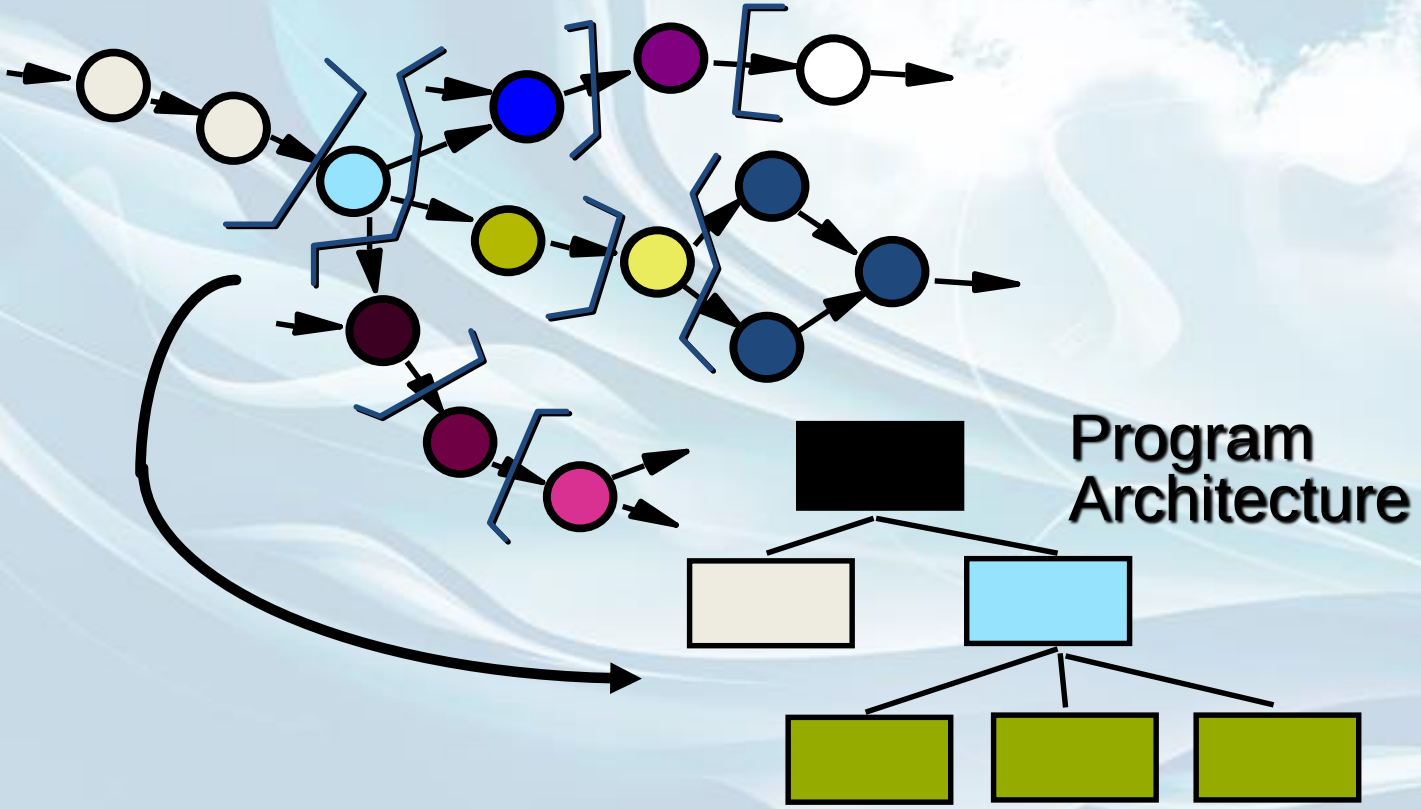
Metode Desain Arsitektur

customer requirements



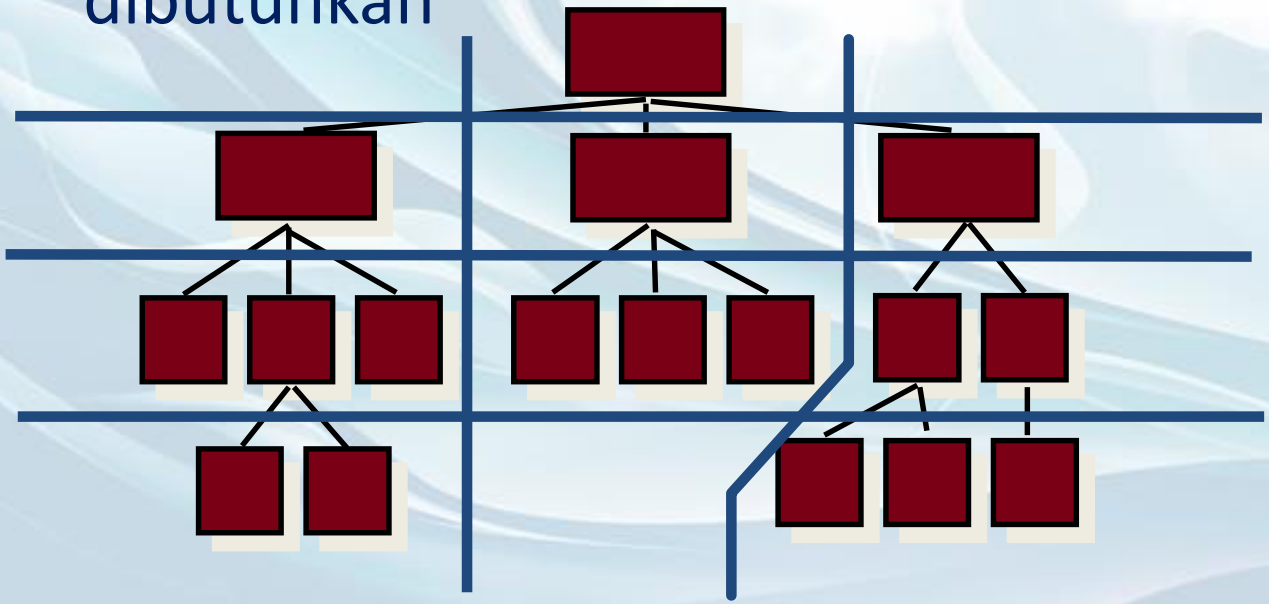
architectural design

Memperoleh Arsitektur Program



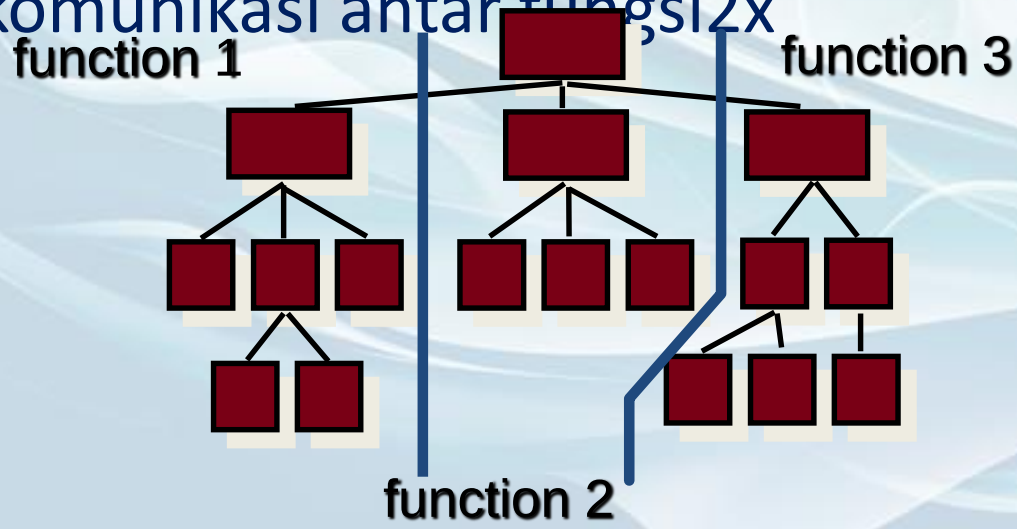
Partisi Arsitektur

- Partisi “horizontal” dan “vertical” dibutuhkan



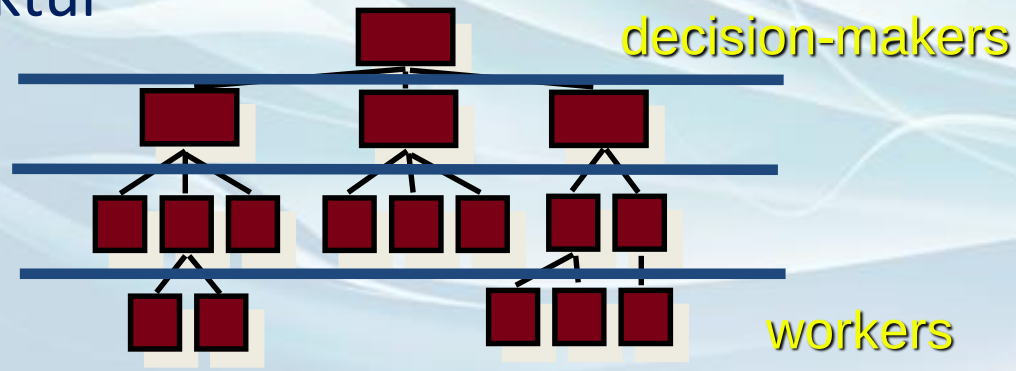
Partisi Horizontal

- Tentukan cabang yang terpisah pada hierarki modul untuk setiap fungsi utama
- Gunakan modul kontrol untuk koodinasi komunikasi antar fungsi2x



Partisi Vertikal : Factoring

- Didesain sehingga pengambilan keputusan dan pekerjaan distratifikasi
- Modul pengambilan keputusan tetap ada di puncak arsitektur



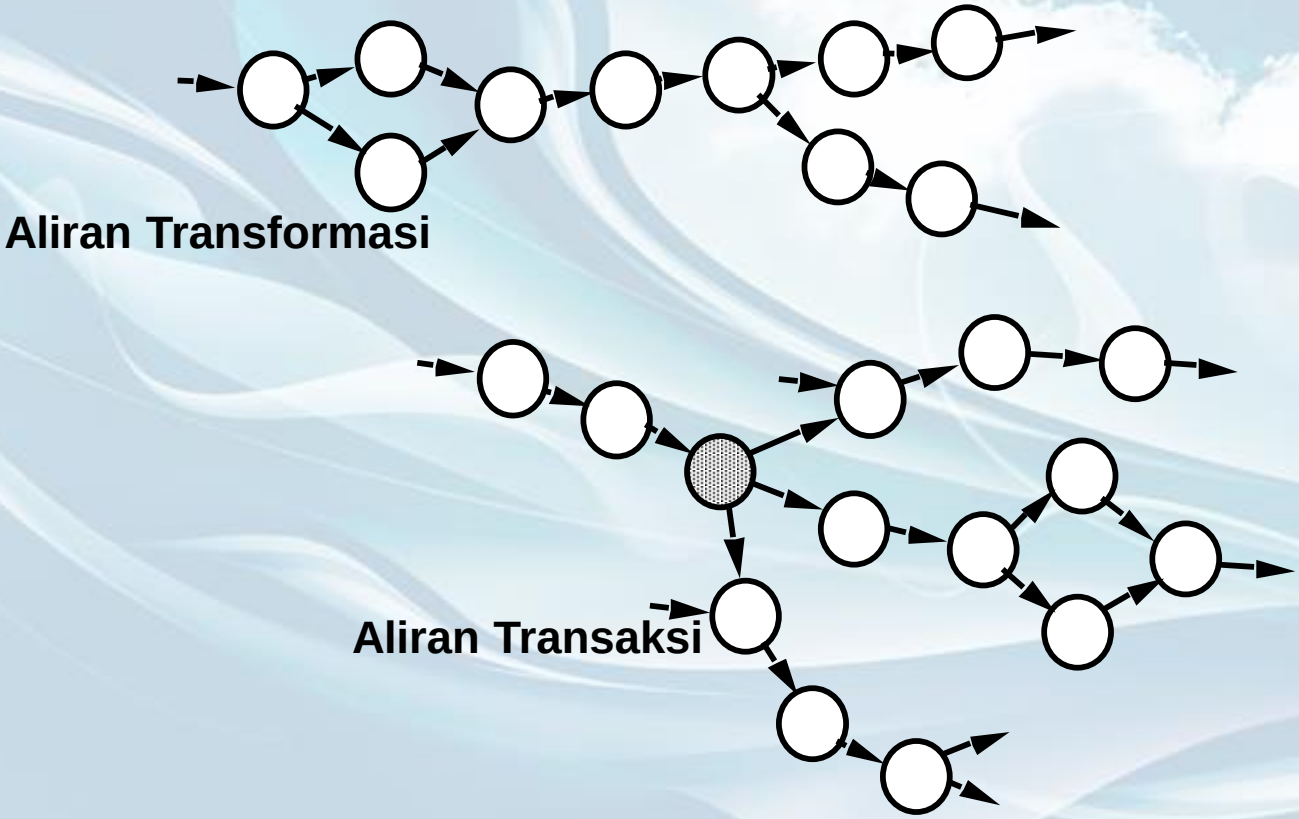
Mengapa Arsitektur Terpartisi?

- Hasilnya adalah PL yang mudah diuji
- Membawa kepada PL yang lebih mudah dikelola
- Hasilnya efek samping yang semakin sedikit
- Hasilnya adalah PL yang lebih mudah dikembangkan

Desain Terstruktur

- Tujuan : untuk mendapatkan arsitektur program yang terpartisi
- pendekatan:
  - DFD dipetakan ke arsitektur program
  - PSPEC dan STD digunakan untuk mengindikasikan setiap modul
- notasi: diagram struktur

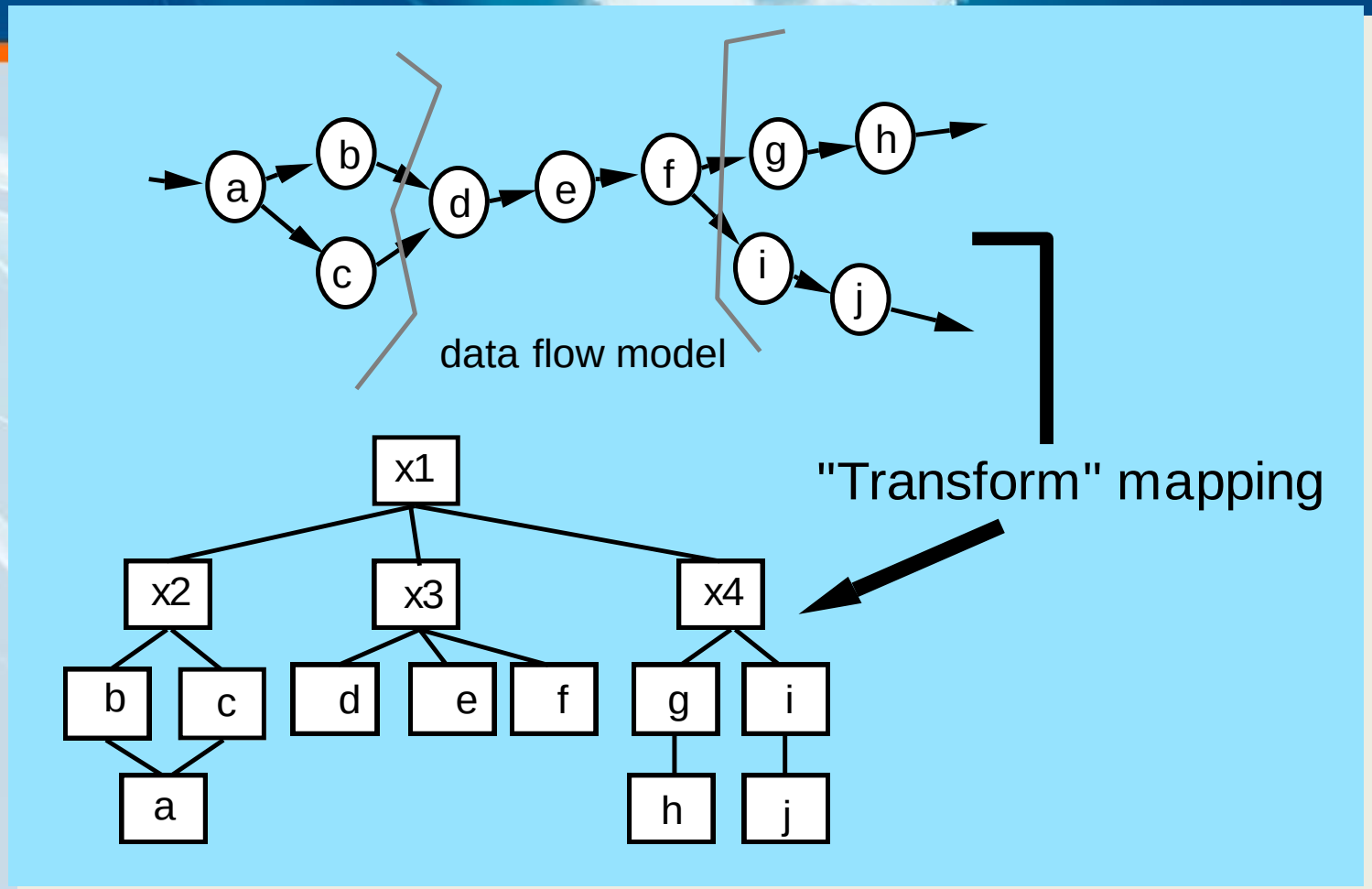
Karakteristik Aliran



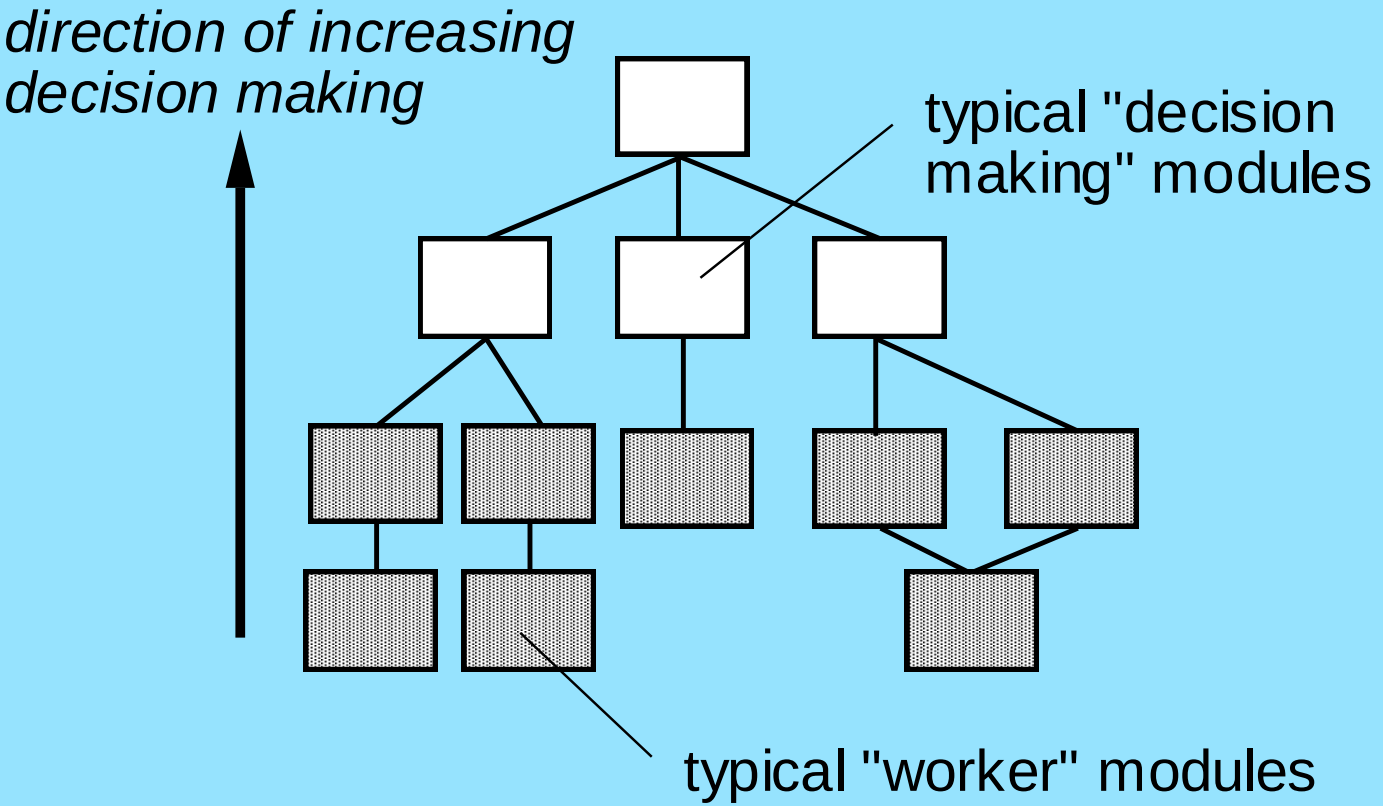
Pendekatan Pemetaan Umum

- Isolasi aliran ke dalam dan ke luar batasan; untuk aliran transaksi, isolasi Pusat transaksi
- Bekerja dari batasan luar, petakan Transformasi DFD ke modul terkait
- Tambahkan modul kontrol jika dibutuhkan
- Sempurnakan struktur program Menggunakan konsep modularitas efektif

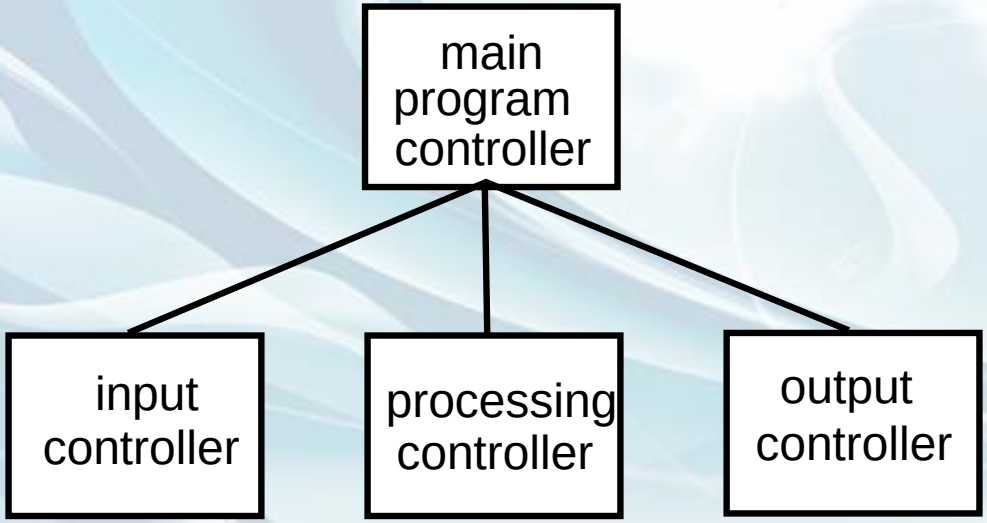
Pemetaan Transformasi



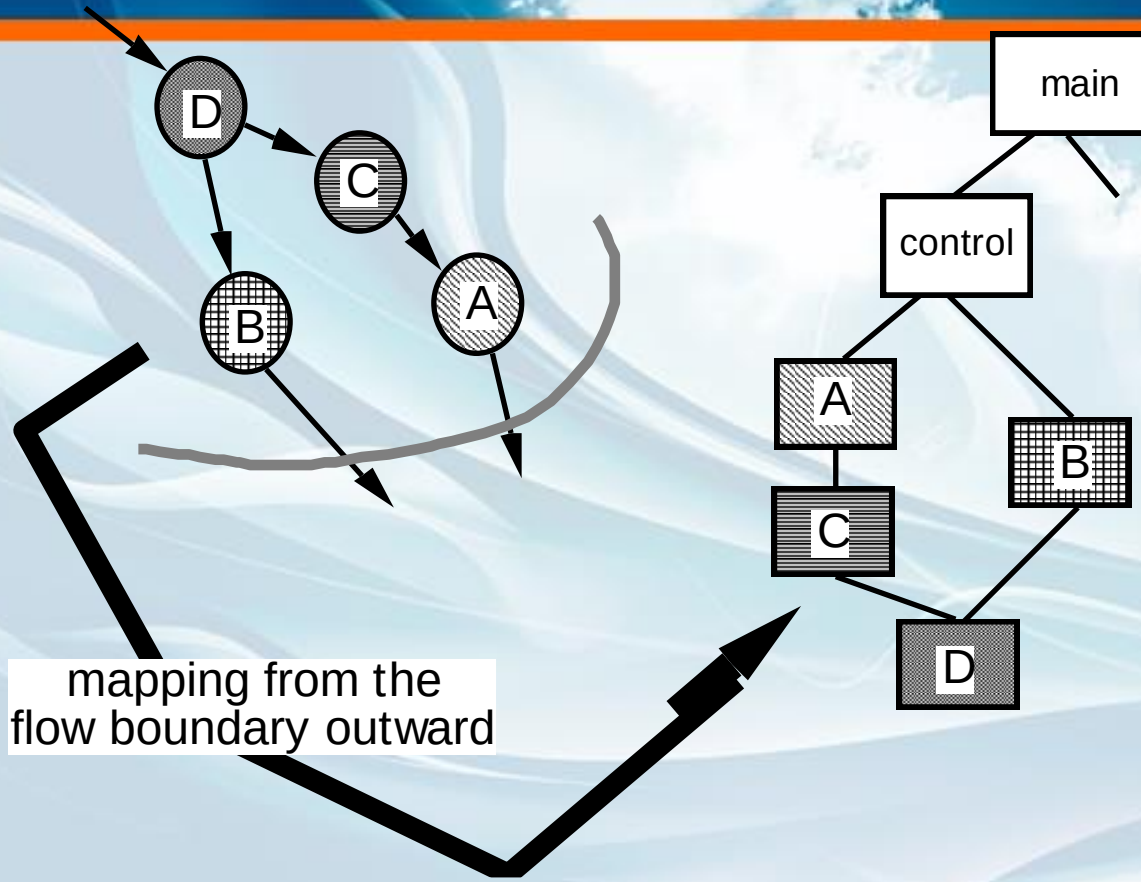
Factoring



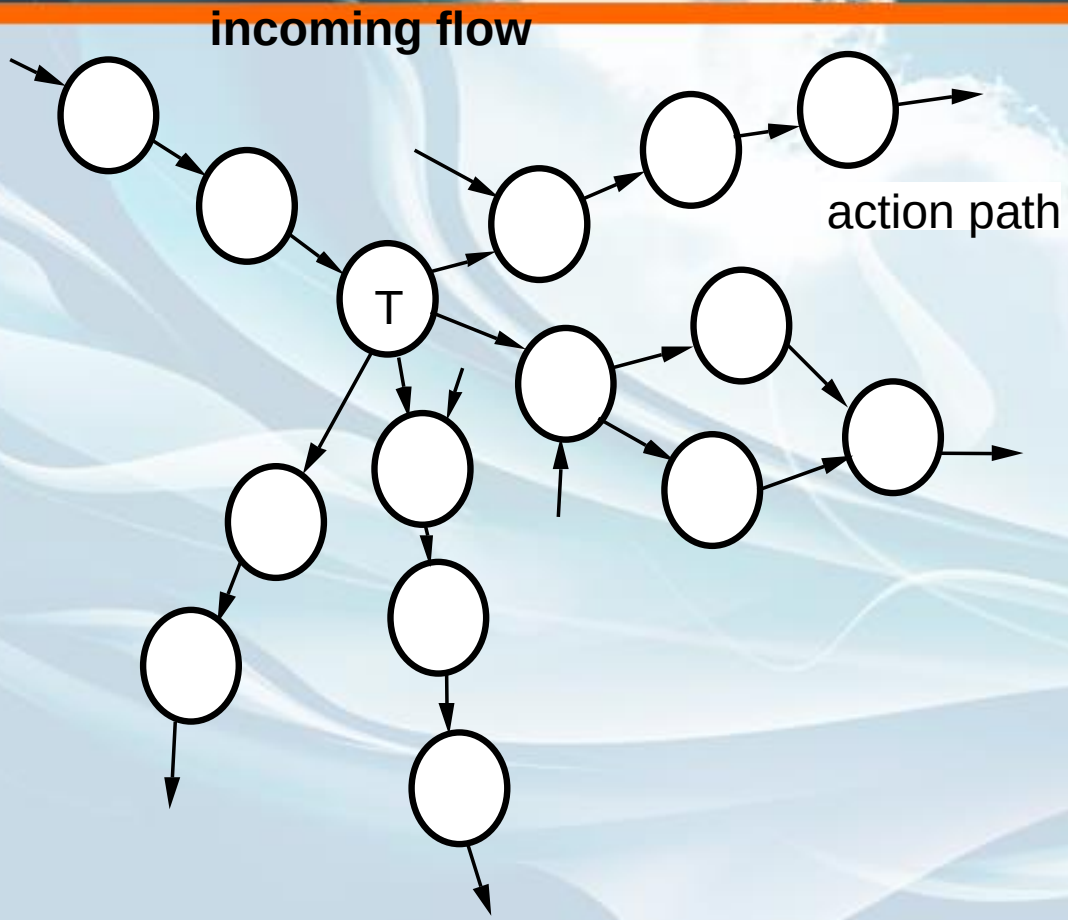
First Level Factoring



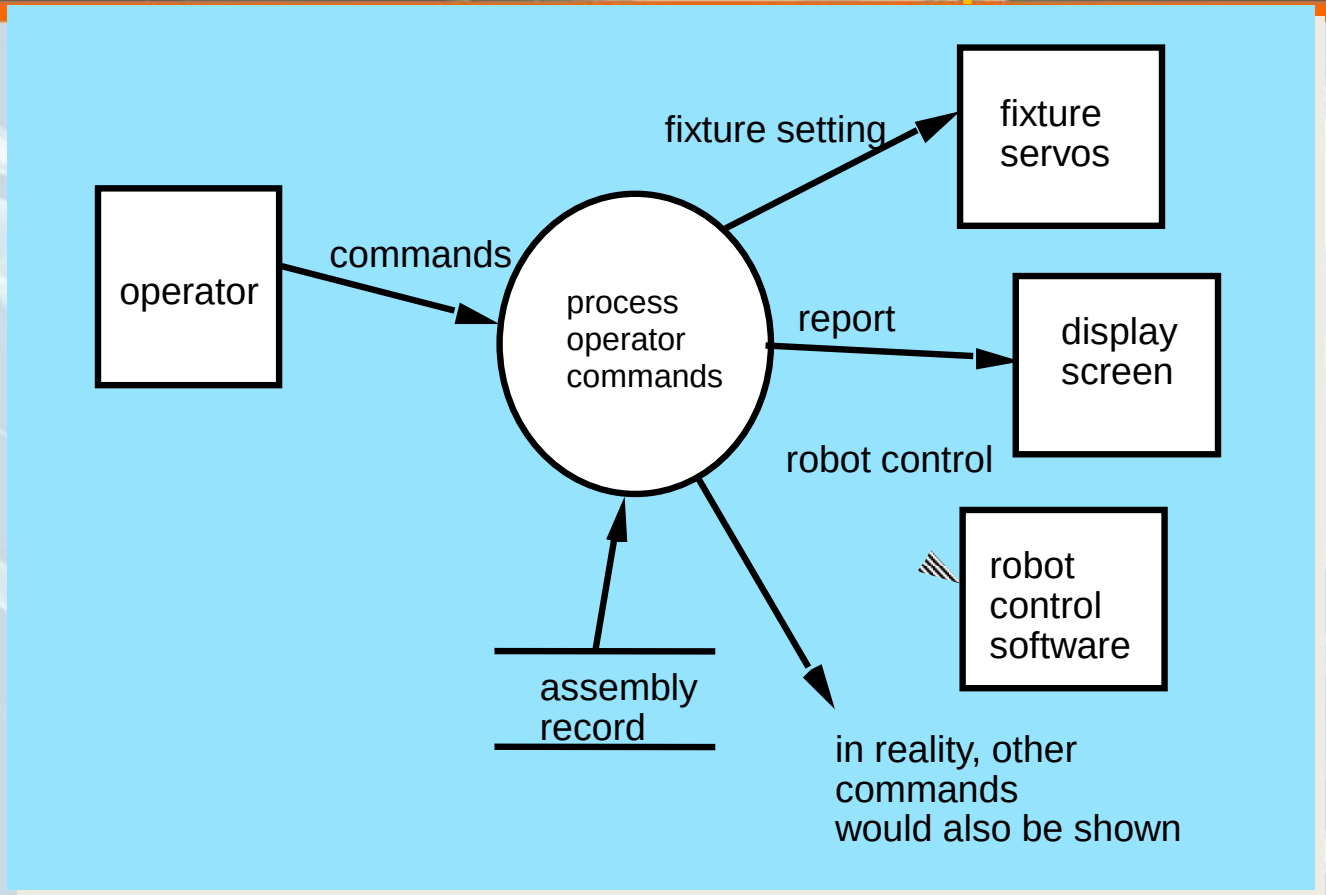
Second Level Mapping



Transaction Flow



Transaction Example



Refining the Analysis Model

1. write an English language processing narrative for the level 01 flow model
2. apply noun/verb parse to isolate processes, data items, store and entities
3. develop level 02 and 03 flow models
4. create corresponding data dictionary entries
5. refine flow models as appropriate

... now, we're ready to begin design!

Deriving Level 1

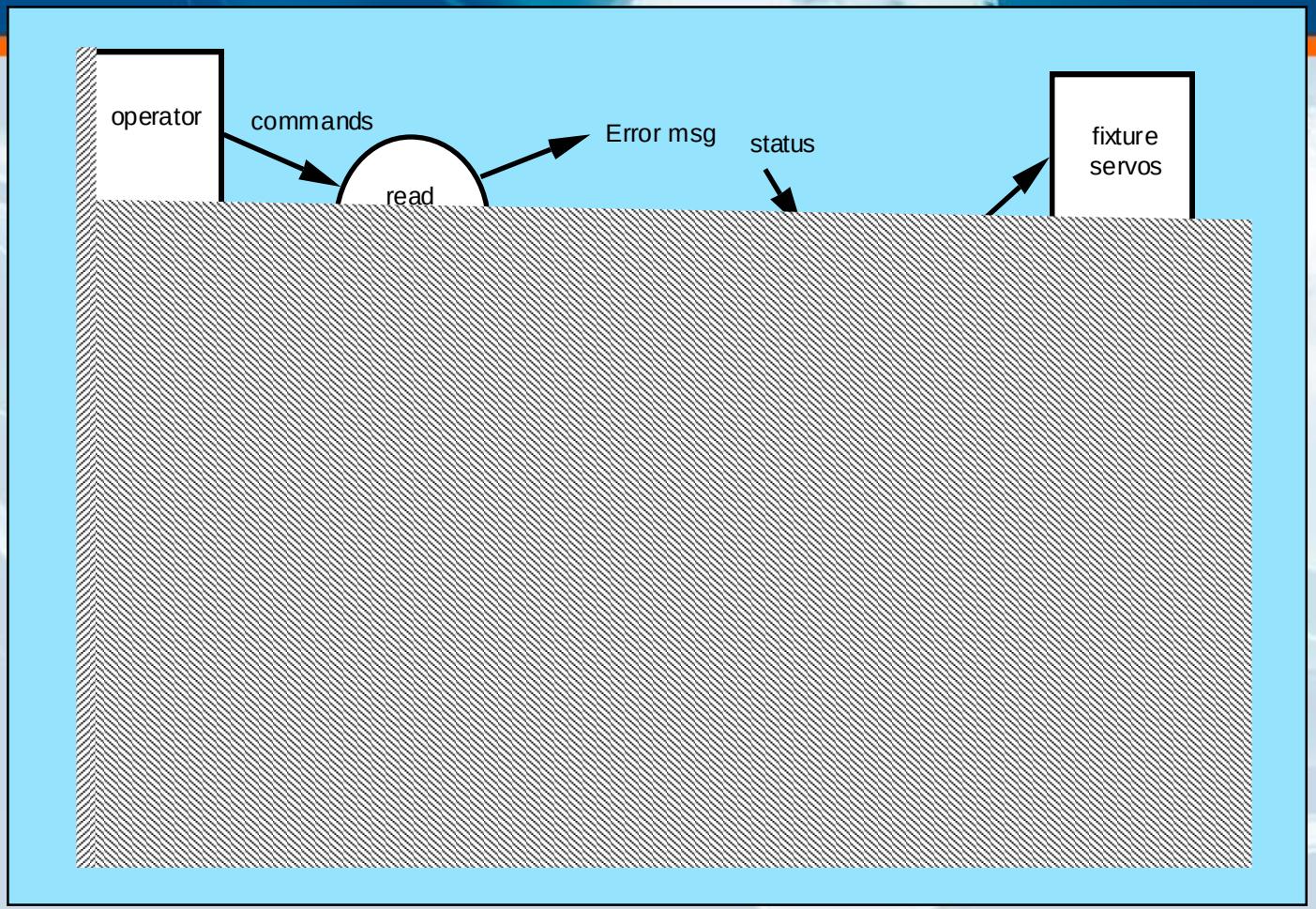
Processing narrative for " process operator commands"

Process operator command software reads operator commands from the cell operator. An error message is displayed for invalid commands. The command type is determined for valid commands and appropriate action is taken. When fixture commands are encountered, fixture status is analyzed and a fixture setting is output to the fixture servos. When a report is selected, the assembly record file is read and a report is generated and displayed on the operator display screen. When robot control switches are selected, control values are sent to the robot control system.

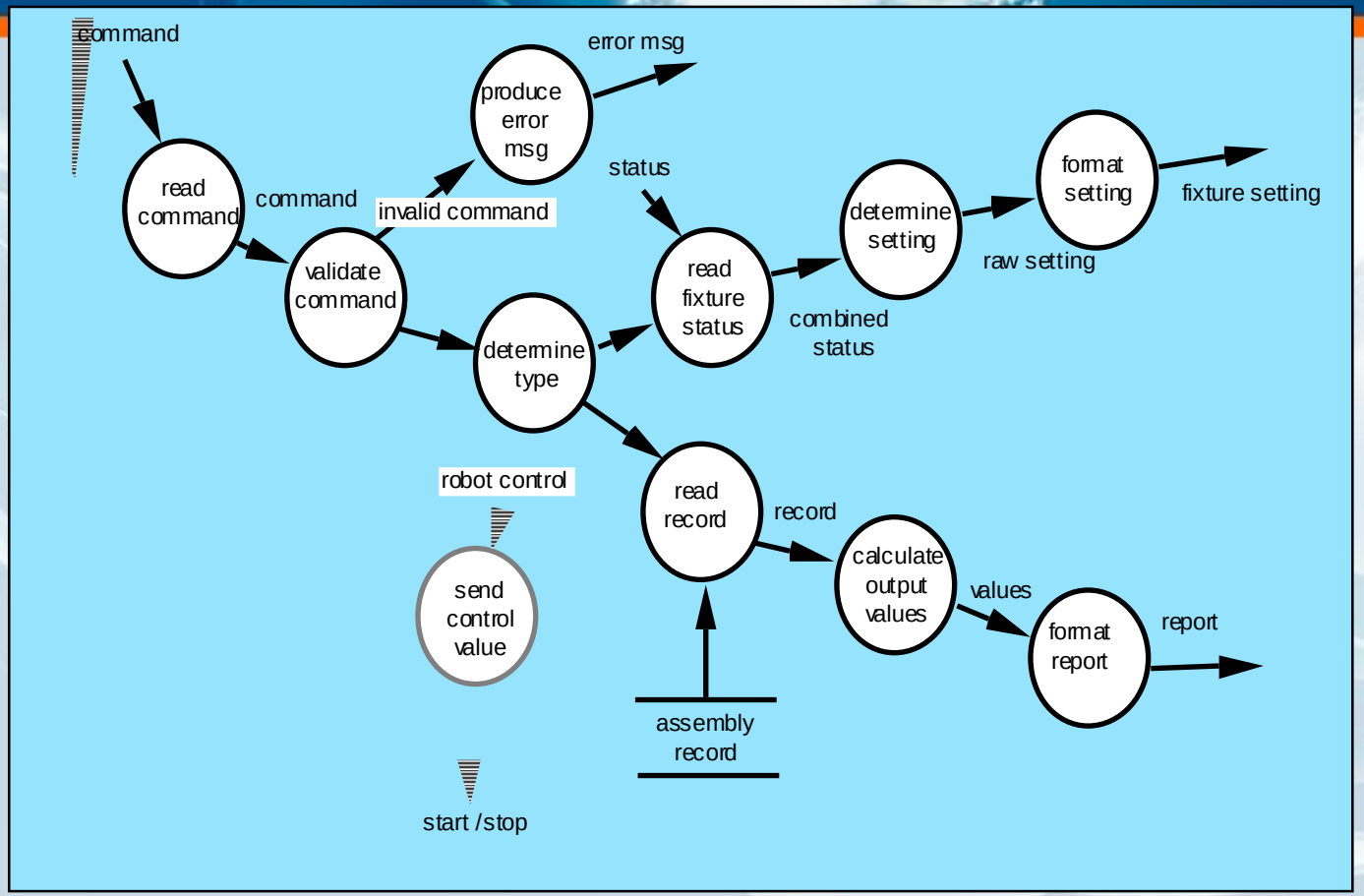
noun-verb parse

Process operator command software reads operator *commands* from the cell *operator* . An *error message* is displayed for *invalid commands* . The *command type* is determined for *valid commands* and appropriate action is taken. When *fixture commands* are encountered, *fixture status* is analyzed and a *fixture setting* is output to the *fixture servos* . When a *report* is selected, the *assembly record file* is read and a report is generated and displayed on the operator *display screen* . When *robot control switches* are selected , *control values* are sent to the *robot control system*.

Level 1 Data Flow Diagram



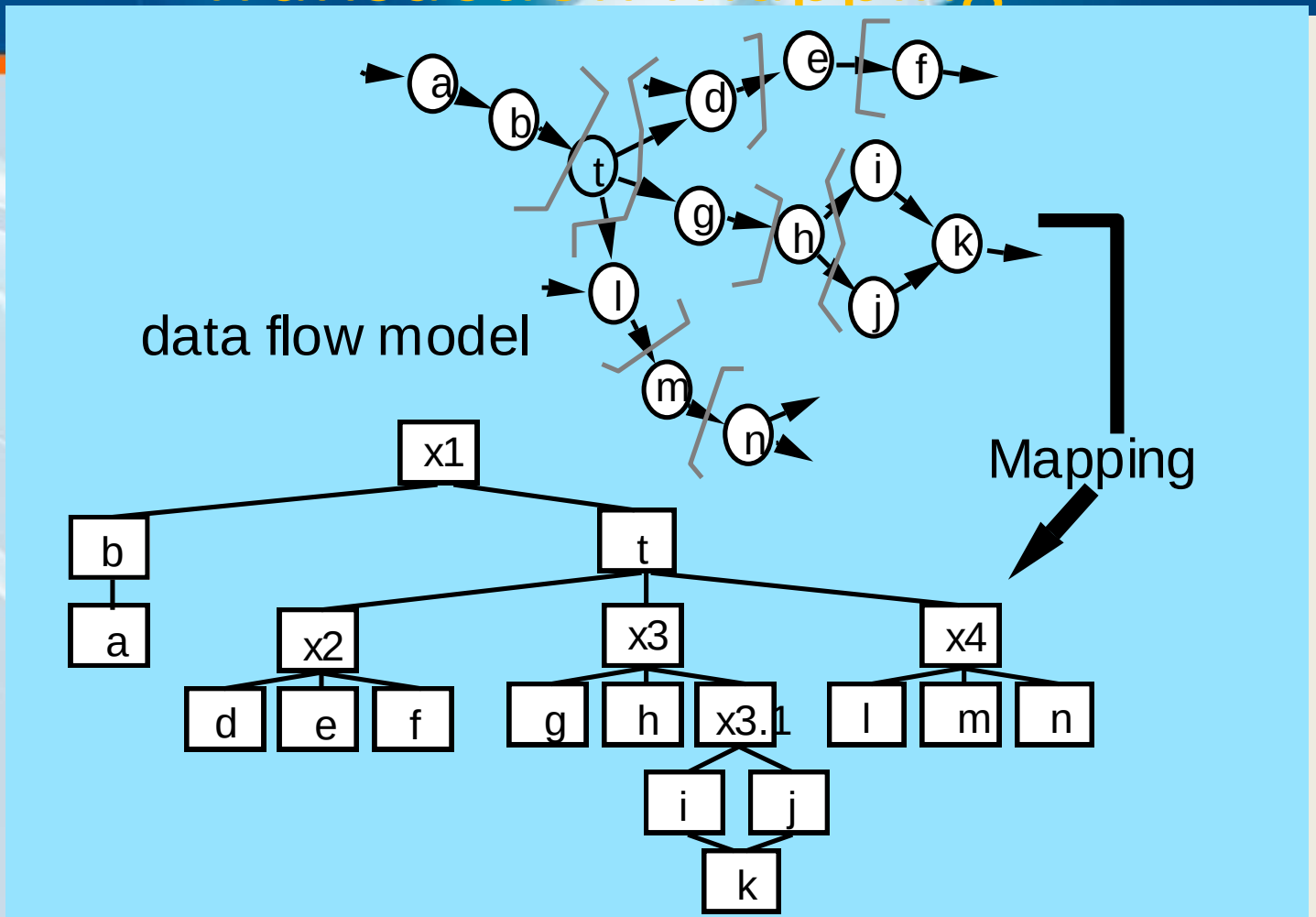
Level 2 Data Flow Diagram



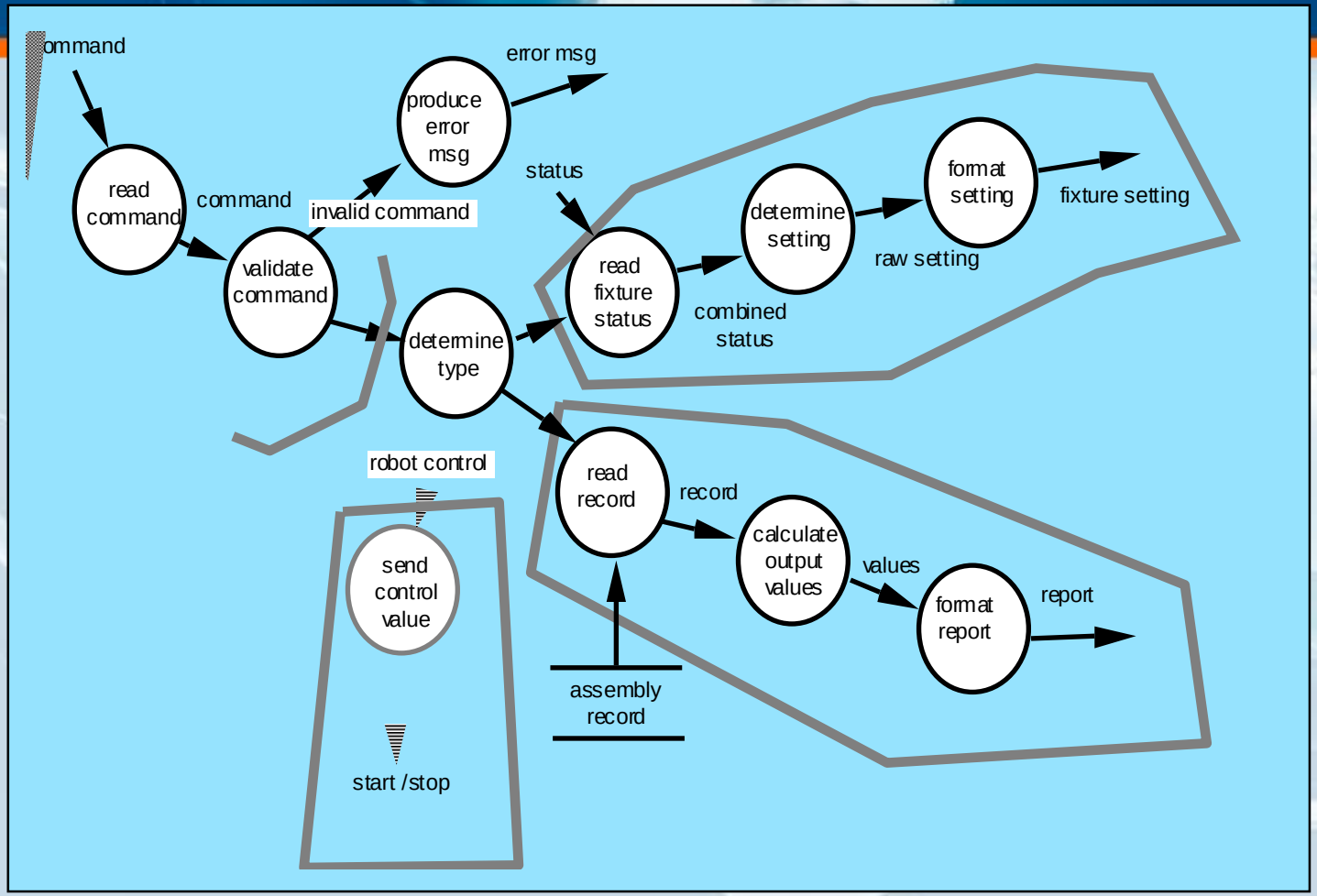
Transaction Mapping Principles

- isolate the incoming flow path
- define each of the action paths by looking for the "spokes of the wheel"
- assess the flow on each action path
- define the dispatch and control structure
- map each action path flow individually

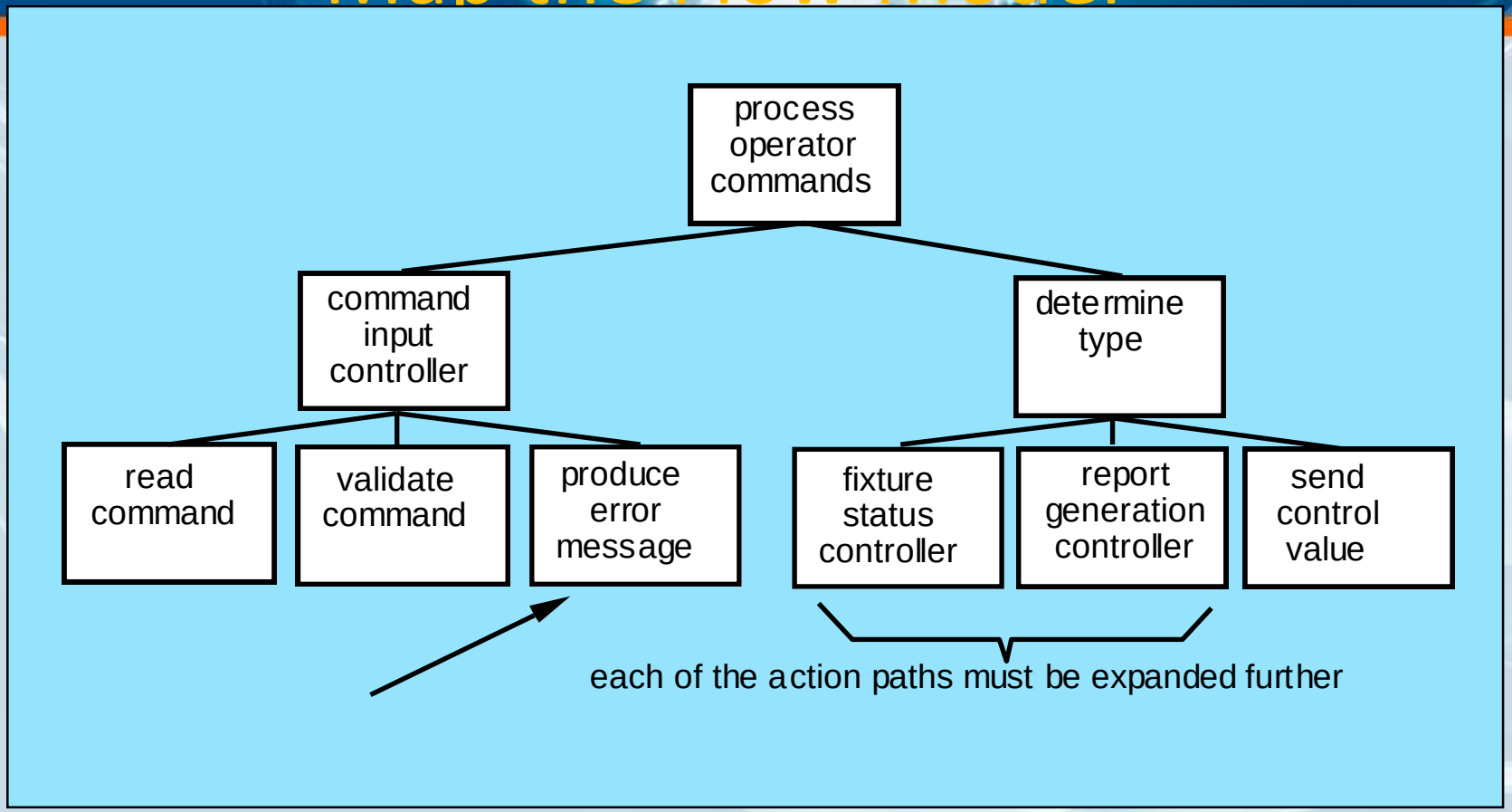
Transaction Mapping



Isolate Flow Paths



Map the Flow Model



Refining the Structure Chart