Catatan Kuliah

Rekayasa Perangkat Lunak (Software Engineering)

Bagian 2

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

Chapter 9
Rekayasa Desain

Konsep Desain OO

- Desain Class
  - Entity classes
  - Boundary classes
  - Controller classes
- Inheritance—semua tanggung jawab superclass akan diwarisi oleh semua subclassnya
- Messages—stimulasi beberapa perilaku yang dapat terjadi pada objek penerima pesan
- Polymorphism—sebuah karakteristik yang mengurangi usaha yang dibutuhkan untuk memperluas desain

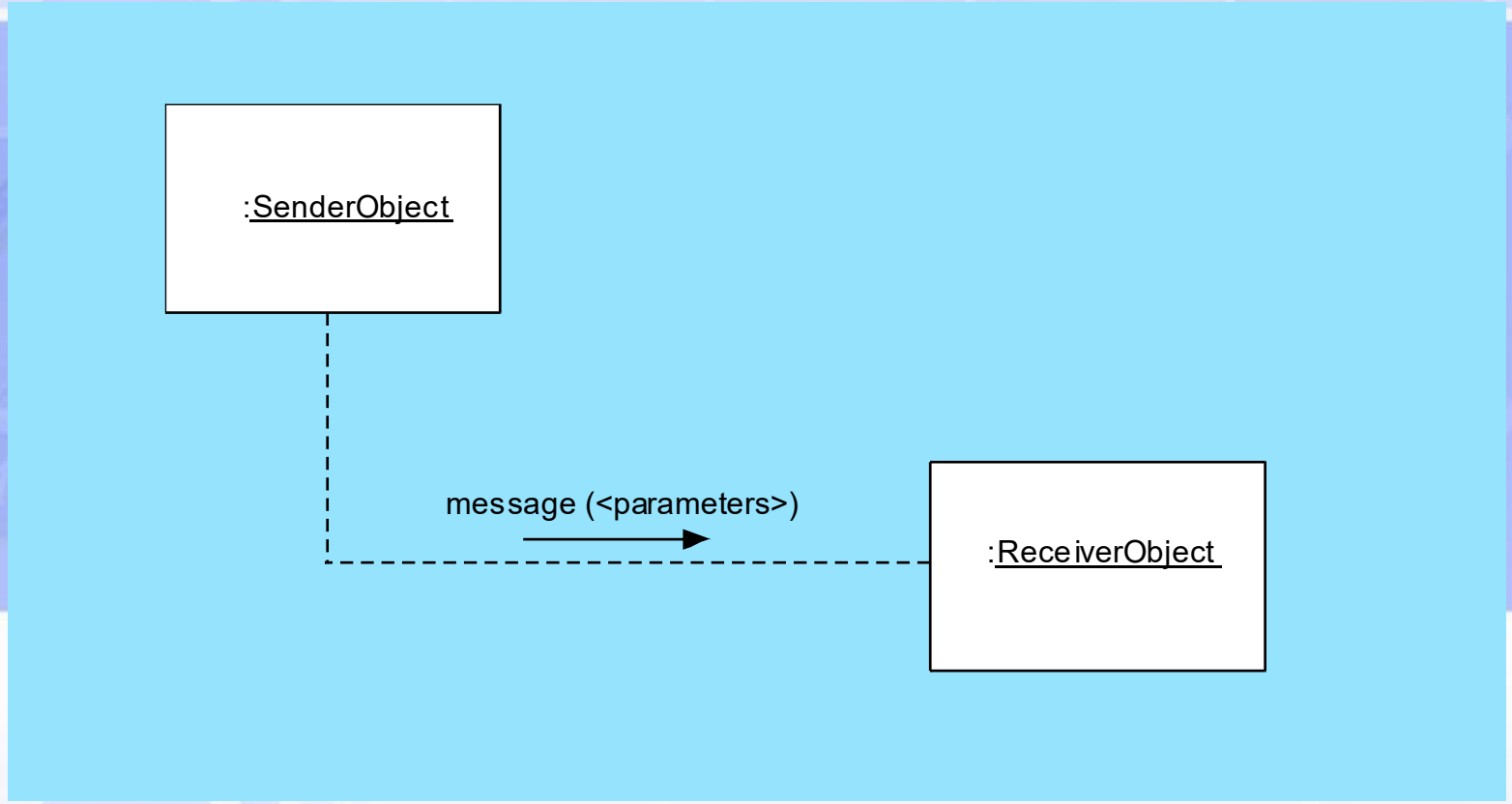
Desain Class

- Analisis class disempurnakan dalam desain untuk menjadi class-class entitas
- Class-class boundary dikembangkan selama desain untuk membuat interface(mis. Layar interaktif atau laporan cetak) yang dilihat pengguna dan berinteraksi.
  - Class-class boundary didesain dengan tanggungjawab untuk mengelola cara objek entitas ditampilkan kepada user.
- Class-class control didesain untuk mengelola :
  - Pembuatan atau perubahan objek entitas;
  - Instansiasi object boundary dengan mengambil informasi dari objek entitas;
  - Komunikasi kompleks antara sekelompok objek;
  - Validasi data yang dikomunikasikan antar objek atau antar pengguna dan aplikasi.

Inheritance

- Pilihan-pilihan desain :
  - Class dapat didesain dan dibangun dari nol. Jika demikian, inheritance tidak digunakan.
  - Hierarki class dapat dicari untuk mencari kemungkinan jika sebuah class yang lebih tinggi pada hierarki (superclass) memiliki atribut-atribut dan operasi-operasi yang paling banyak dibutuhkan. Class baru ini diturunkan dari superclass dan beberapa tambahan dapat diberikan jika dibutuhkan.
  - Hierarki class dapat di restrukturisasi sehingga atribut-atribut dan operasi-operasi yang dibutuhkan dan diturunkan oleh class baru tersebut.
  - Karakteristik dari class yang sudah ada dapat di-override dan atribut-atribut dan operasi-operasi dengan versi berbeda dapat diimplementasikan untuk class baru tersebut.

Messages



Polymorphism

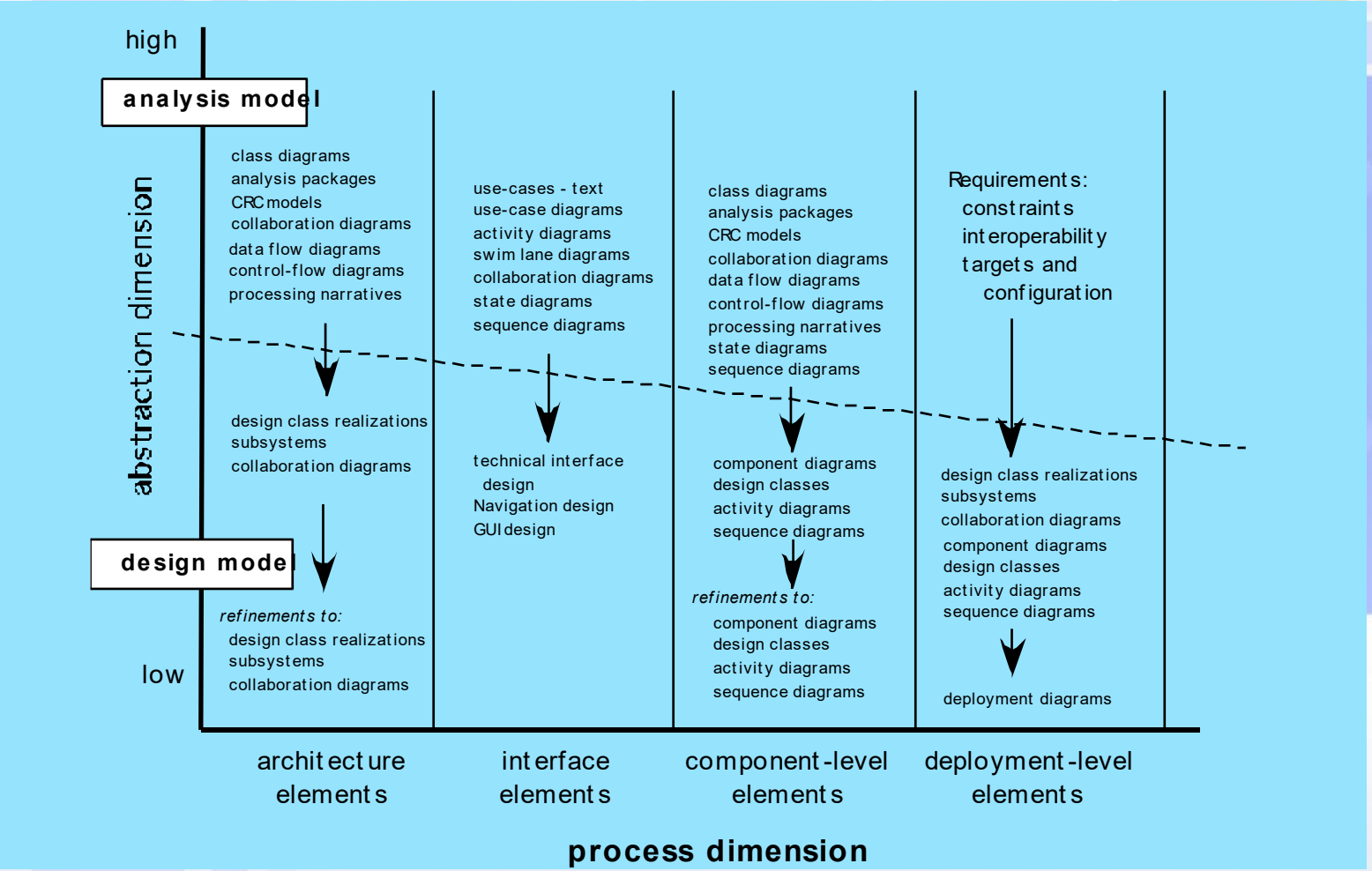
Pendekatan konvensional …

```
case of graphtype:
        if graphtype = linegraph then DrawLineGraph (data);
        if graphtype = piechart then DrawPieChart (data);
        if graphtype = histogram then DrawHisto (data);
        if graphtype = kiviat then DrawKiviat (data);
end case;
```

Semua graphs menjadi subclass dari class umum yang disebut graph. Menggunakan konsep overloading, setiap subclass mendefinisikan operasi yang disebut *draw*. Sebuah object dapat mengirim pesan draw pada salah satu instansiasi objek dari salahsatu subclassnya. Objek yang menerima message akan menjalankan operasi draw-nya sendiri untuk membuat graph yang sesuai..

```
graphtype draw
```

Model Desain



Elemen-Elemen Model Desain

- Elemen-elemen Data
  - Data model --> struktur data
  - Data model --> arsitektur database
- Elemen-elemen arsitektur
  - Domain aplikasi
  - Class-class analisis, relasinya, kolaborasi dan perilaku diubah menjadi realisasi desain
  - Patterns dam “styles” (Chapter 10)
- Elemen-elemen interface
  - user interface (UI)
  - Interface external pada sistem lain, piranti-piranti, jaringan-jaringan atau produsen maupun konsumen informasi lainnya
  - Interface internal antara komponen-komponen desain.
- Elemen-elemen komponen
- Elemen-elemen deploy

Elemen-elemen Interface

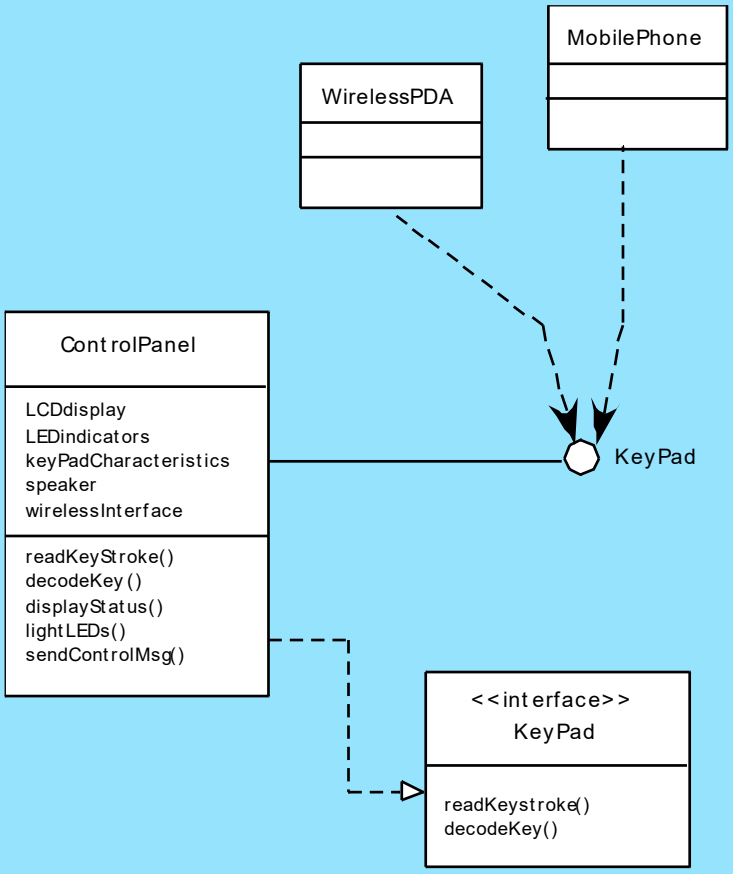
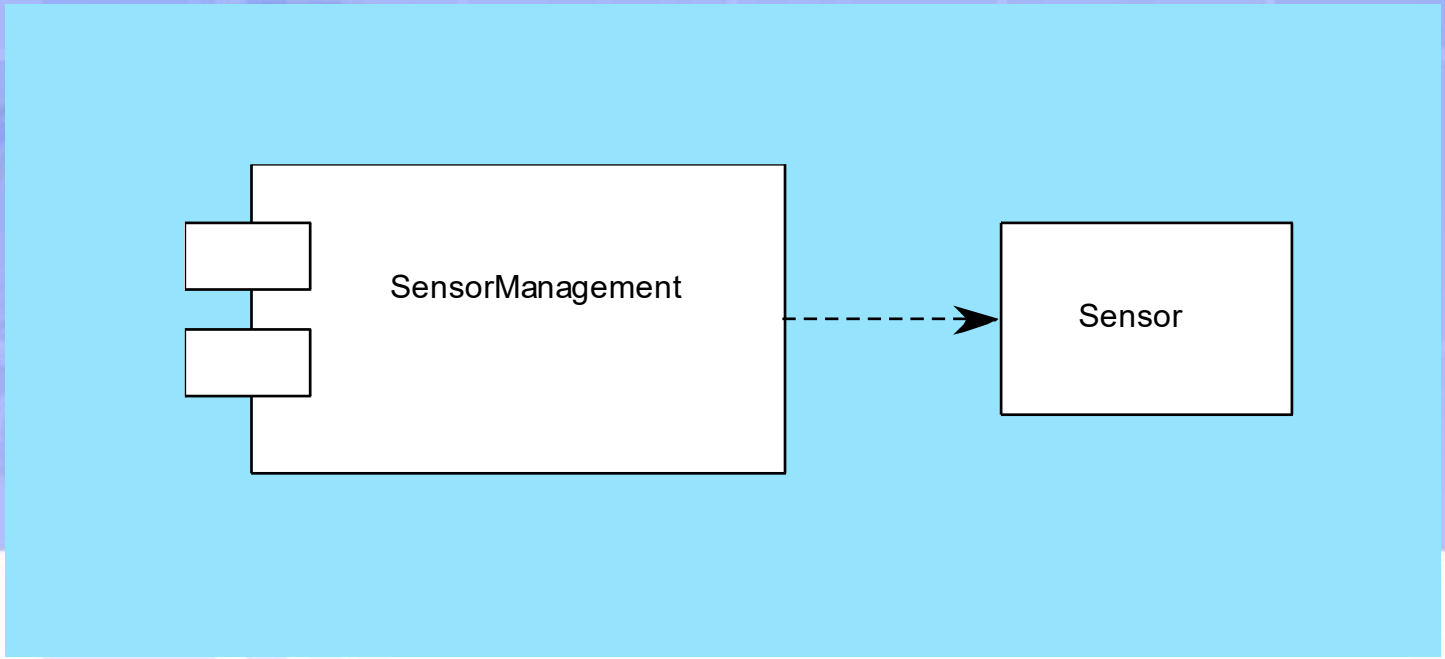


Figure 9.6 UML interface representation for **ControlPanel**

Elemen-elemen Komponen



Elemen-elemen Deployment

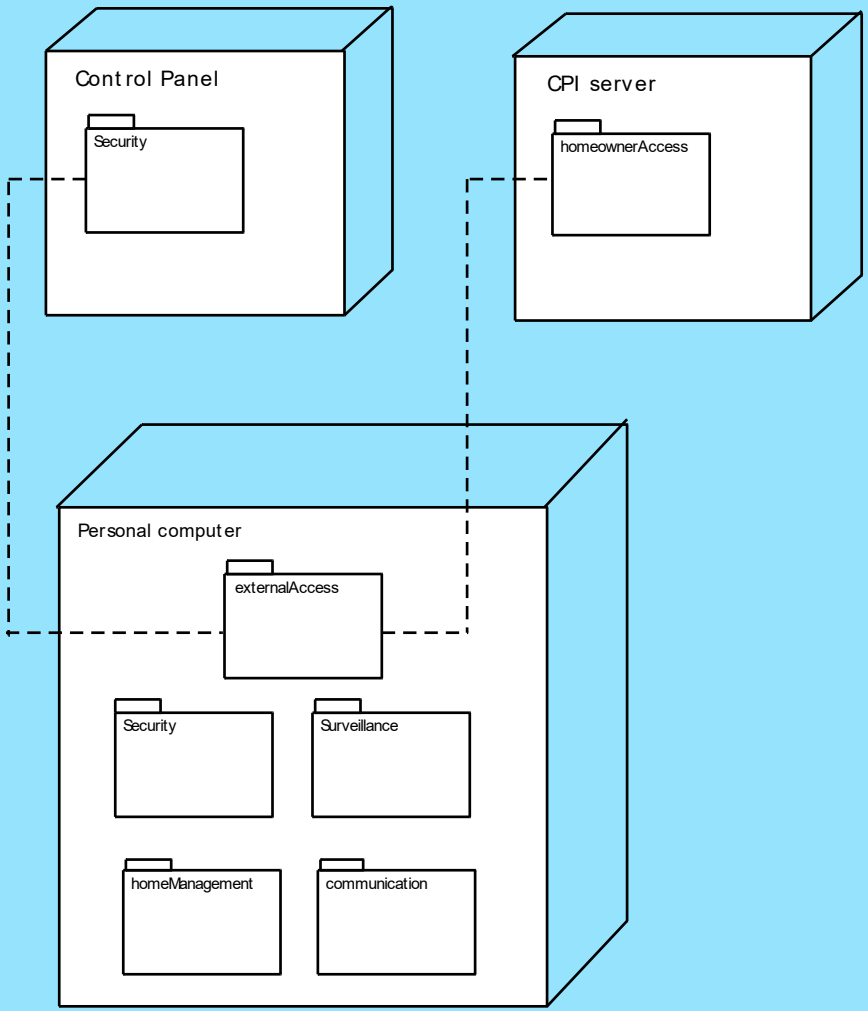


Figure 9.8 UML deployment diagram for *SafeHome*

Design Patterns

- Desainer terbaik di segala bidang tetap mempunyai keterbatasan untuk melihat pola yang mencirikan sebuah masalah dan menghubungkannya dengan pola yang dapat dikombinasikan untuk membuat solusi
- Sebuah deskripsi dari design pattern dapat juga dilihat sebagai sekumpulan design forces.
  - *Design forces* menjelaskan kebutuhan non fungsional (misalkan : kemudahan perawatan, portabilitas) yang dihubungkan dengan PL dimana pattern akan diaplikasikan.
- Karakteristik pattern (class, tanggungjawab, dan kolaborasi) mengindikasikan atribut-atrobit desain yang harus diatur untuk memungkinkan pattern mengakomodasi permasalahan yang bervariasi.

Frameworks

- Sebuah framework bukan merupakan pattern arsitektur, namun lebih merupakan kerangka dengan sekumpulan “plug points” (yang juga disebut *hooks* dan *slots*) yang memungkinkannya untuk beradaptasi dengan domain permasalahan tertentu.
- Gamma et al mencatat bahwa:
  - Design patterns lebih abstrak dari frameworks.
  - Design patterns adalah elemen-elemen arsitektural yang lebih kecil daripada frameworks
  - Design patterns lebih umum daripada frameworks