

Logical Architecture

Object Oriented Analysis and Design

Aron Trauring

T++ Technical Skills Training Program

CUNY Institute for Software Design & Development (CISDD)

New York Software Industry Association (NYSIA)

December 13th, 2004

Logical Architecture — What?

- Large-scale organization of software classes into packages, subsystems and layers
- Logical — not related to actual deployment
- Big Picture
- Transition from analysis to design
- Not a deployment view (e.g. don't indicate MySQL or Solaris)
- Larman p. 199 — Swing — bad

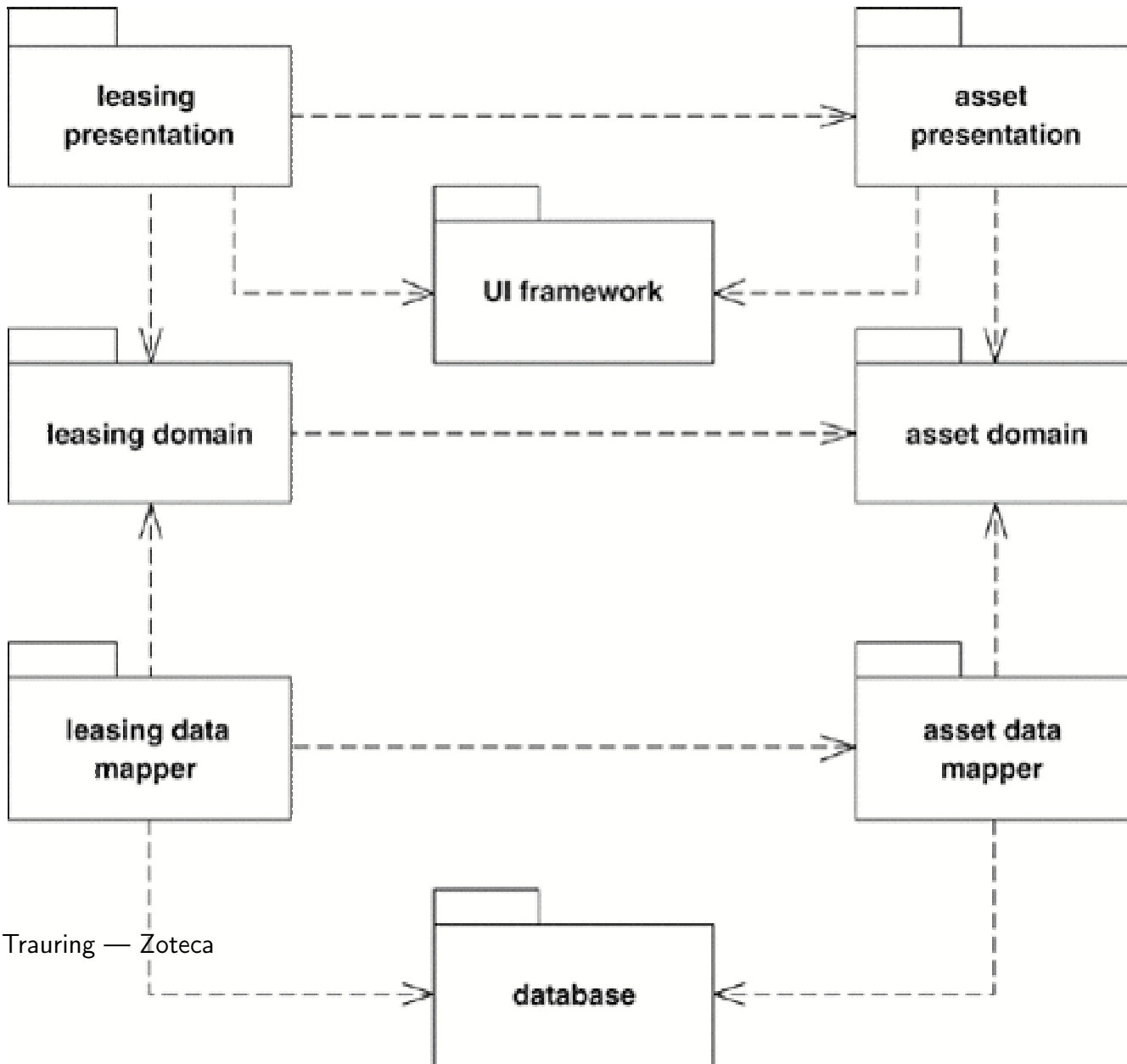
Logical Architecture — How Much?

- Process specific
- Top-down (UP) puts emphasis on up-front effort in architectural design early iterations build core architecture
- Bottom-up (XP) architecture is empirical result of effort — don't design for the future

Grouping Types

- Package — Namespace — Register.Record
- Subsystem — group of packages forming a discrete engine with cohesive responsibilities — Reports
- Layer — major subsystems — UI
- Each group should be internally cohesive and weakly coupled with outside

UML Package Diagram

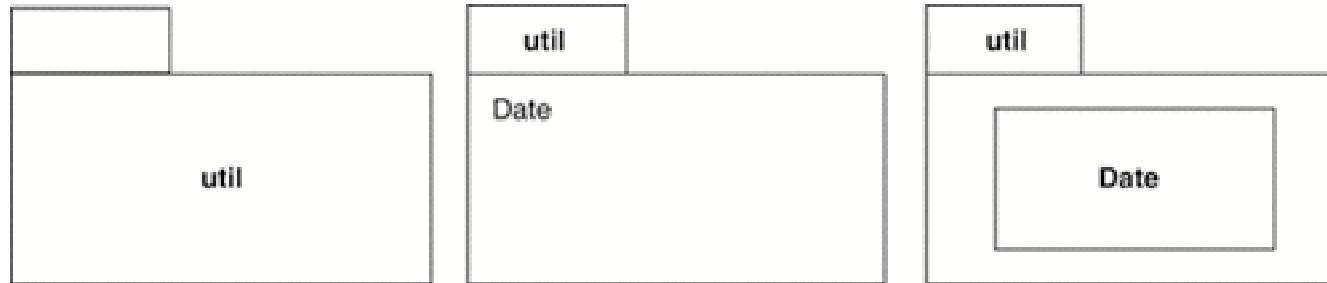


T++ — CISDD — NYSIA

Logical Architecture - OOAD

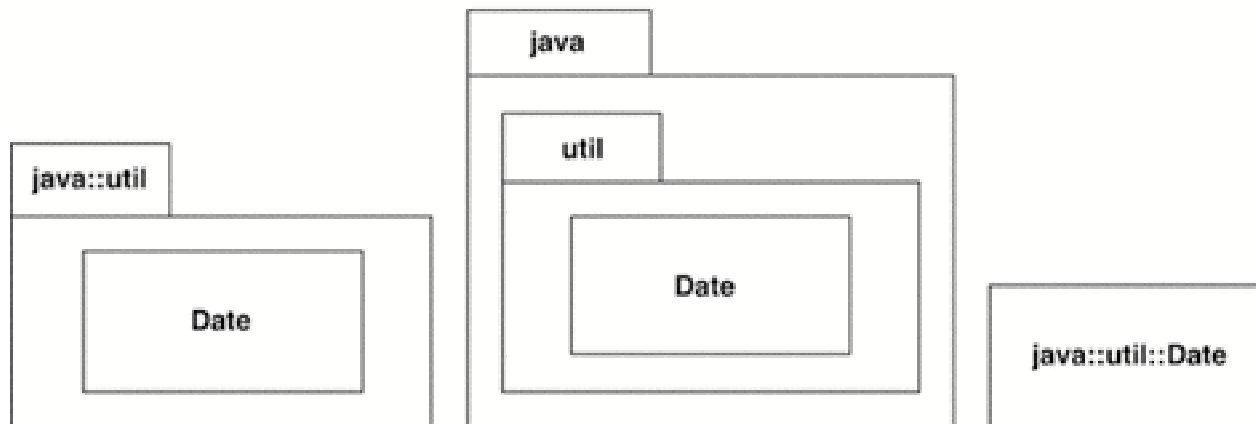
- Package
- Dependency line — large scale coupling

Packages as Namespaces



Contents listed in box

Contents diagrammed in box



● Fully qualified package name

Nested packages

Fully qualified class name

Package Diagrams — When?

- UP — design up front sketches, and then generate backwards to keep model up to date
- XP — generate backwards from code to help in re-organization
- Particularly useful for mapping dependencies in large scale systems

Principles for Grouping Classes in Packages

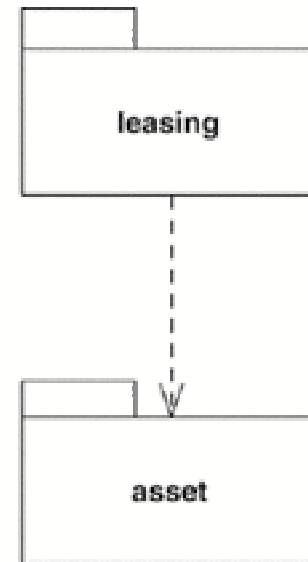
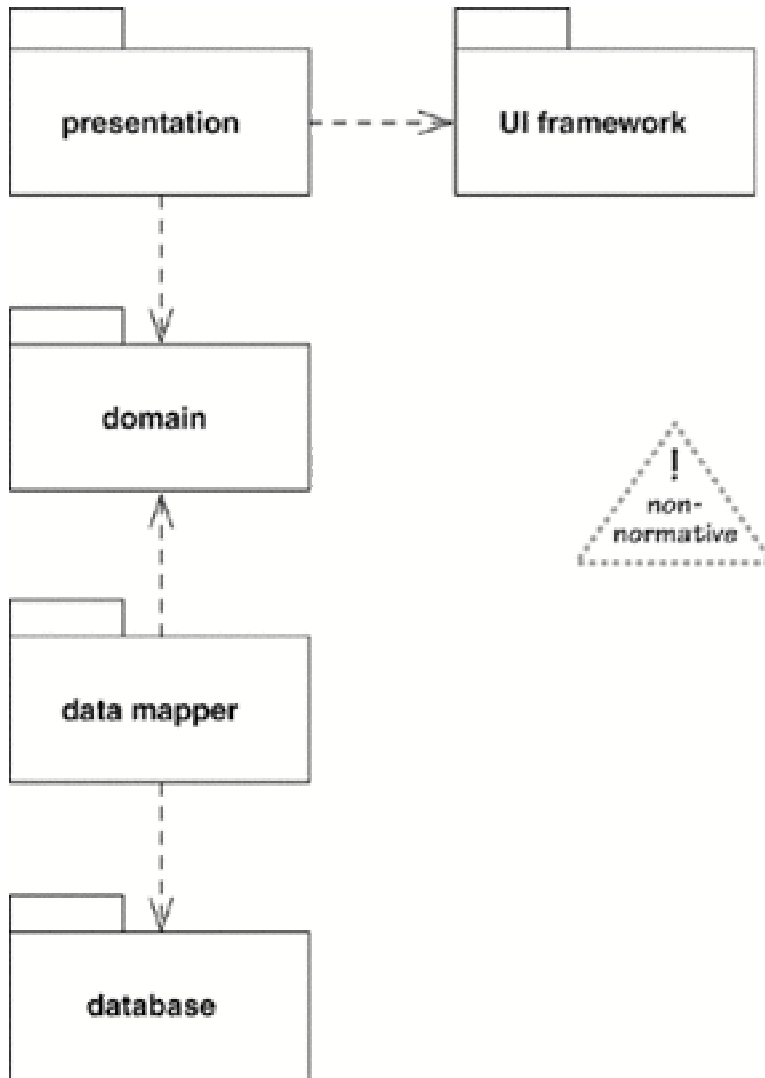
- *Common Closure Principle* — classes in a package should need changing for similar reasons. The
- *Common Reuse Principle* — classes in a package should all be reused together

Clear Flow of Dependency [Martin]

- Flow in one direction (with well-defined exceptions)
- *Acyclic Dependency Principle* — limit cycles in the dependencies — cycles should be localized
- *Stable Dependencies Principle* — the more dependencies coming into a package, the more stable the package's interface needs to be (changes have greater impact),
- *Stable Abstractions Principle* — more stable packages tend to have a higher proportion of interfaces and abstract classes
- Dependency relationships are not transitive — Asset Domain changes effects Leasing Domain not Leasing Presentation

- Packages are used in many places — use a keyword, such as «global»
- General dependency notation enough

Package Diagrams to Clarify Complexity

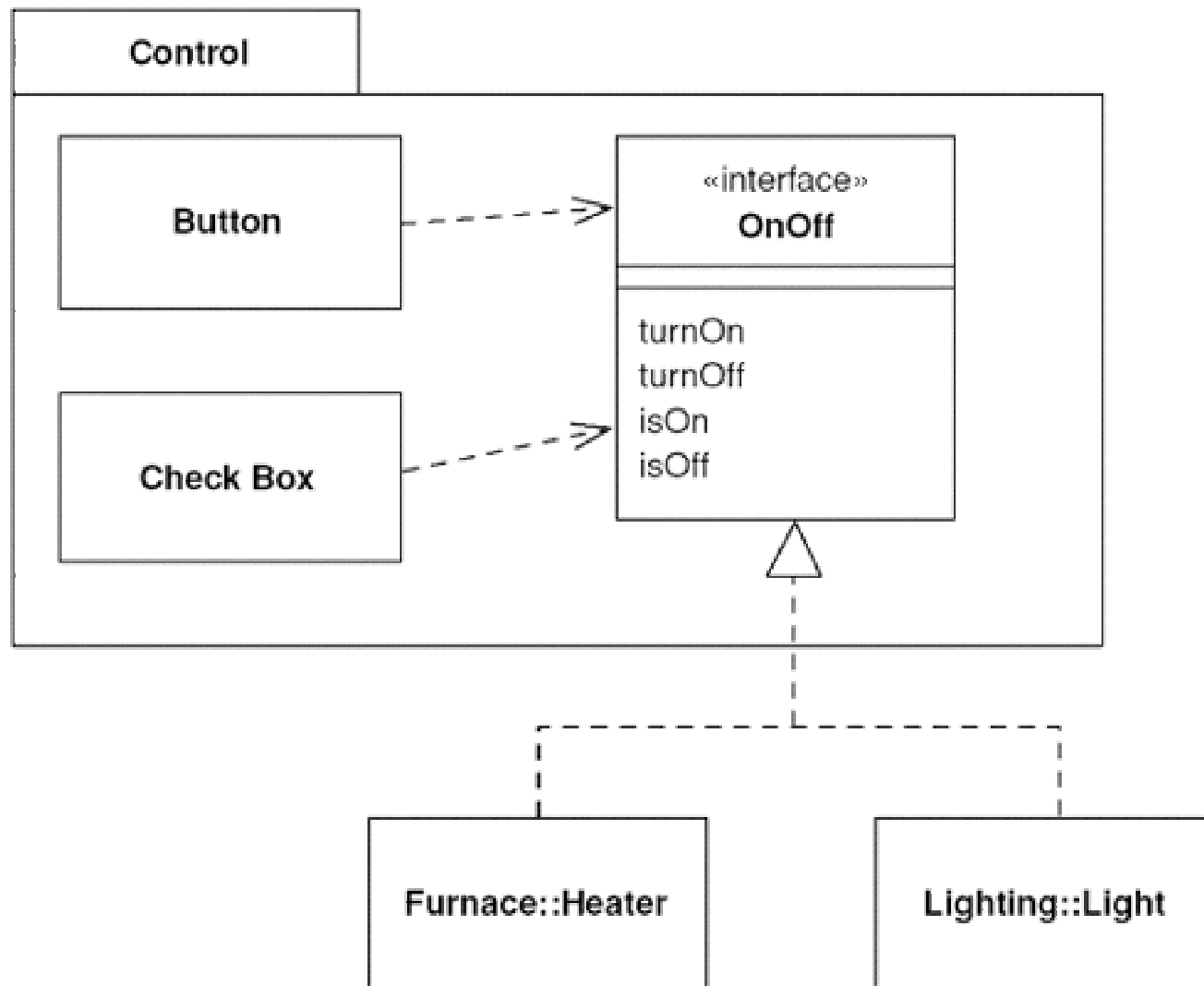


T++ — CISDD — NYSIA

Logical Architecture - OOAD

- Layers aspects
- Subject area aspect

Separating Interface and Implementation



- Delegation / Realization
- Generalization arrow — Inheritance / sub-classing

Layers

- Larman Figure 13.4 p.203

Layers — What?

- Large-scale logical structure of distinct, related responsibilities
- Clean, cohesive separation of concerns
- Lower layers more general
- Higher layers more application specific
- Coupling and collaboration is from higher to lower layers

Layers — Why?

- Low coupling avoids changes rippling through system
- Separation of concerns allows for more flexibility — changing UI or changing DBMS
- Separation of concerns allow for reuse of packages in other systems (particularly lower layers)
- Easier division of work among teams

Model-View Separation Principle

- Do not connect or couple UI objects to non-UI objects
- Do not put application logic in the UI object methods
- Allows for changing views or having multiple views
- allows for no-gui call to application logic