Chapter 2
Theory of Components

Major Topics

- Discuss principles of COP and its major features
- Investigate the infrastructures of COP technologies
- Provide a framework to unify various component technologies
- Formal definitions of software components
- Formal definitions of connections for component assembling
- Formal definitions of component deployment

Principles

- Components represent decomposition and abstraction.
- Reusability should be achieved at various levels.
- CBSD increases the software dependability.
- DBSD could increase the software productivity.
- CBSD promotes software standardization.

Philosophy

- Software components are associated with their component infrastructure.
- Different component technologies have different component infrastructures, and thus have different component definitions.

Infrastructure

- The basic, underlying framework or features of a system or organization
- The fundamental facilities serving a country , a city, or area, as transportation and communication systems.
- What is a component infrastructure?

Component infrastructure (1)

- Component definition revisit – reusable, self-contained, independently deployable software units.
- Component Infrastructure: The underlying foundation or basic framework to construct, assemble, and deploy components.
- Components do not exist without infrastructure.

Component Infrastructure (2)

- CI = (Comp, Conn, Depl)
- Comp = component model
  - What is a component in the CI?
- Conn = connection model
  - How to generate a new component via the existing ones?
- Depl = deployment model
  - How to put components into applications?

Formal Definition

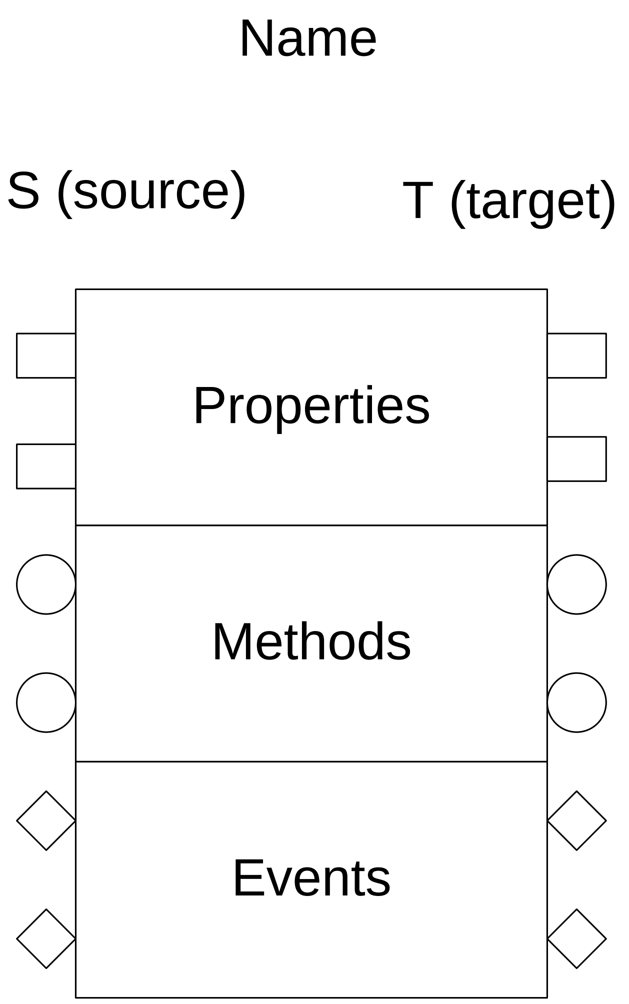
- A component C is a quadruple

  C = (P, M, E, I)

  Where

  - P = { p : T | p is an identifier, T is a type }
  - M = { (v, a, t, i (p)) | v € visibility, a € access, t € type, i € identifier, p € parameter }
  - E = { e : T | e is an event, T is a type }
  - I $\subseteq$ $2^P \times 2^M \times 2^E$

A Component Chart



Why do we need Interface?

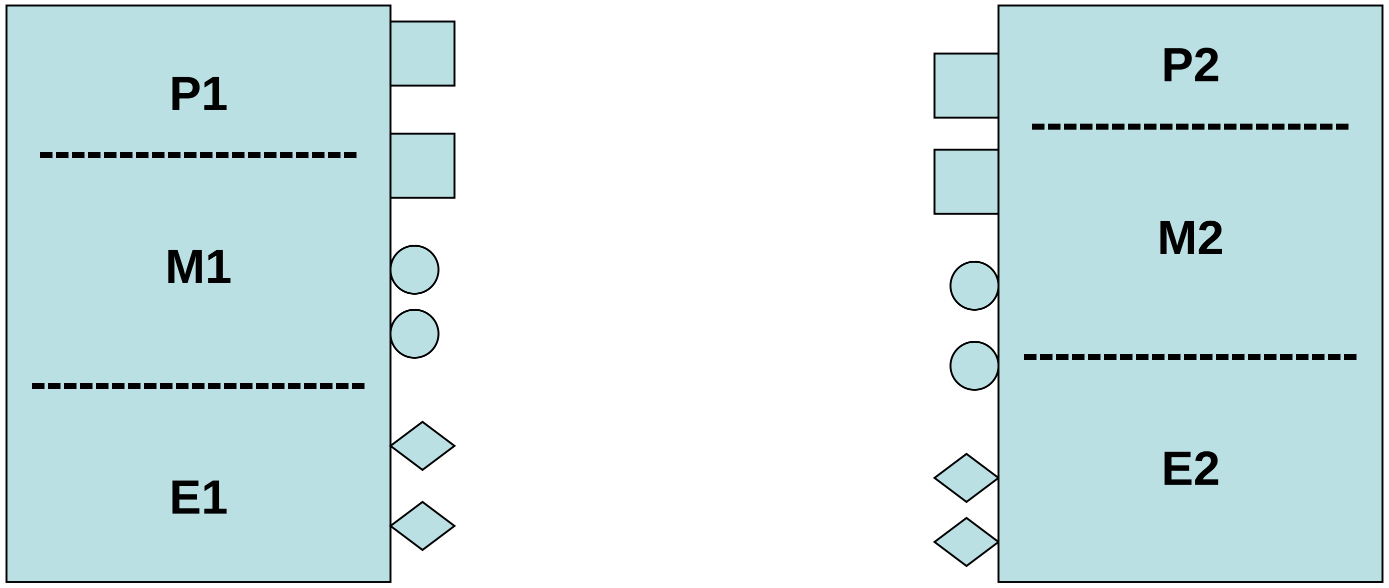
- $C = (P, M, E, I)$
- (P, M, E) part answers the question: *What does this component do?*
- The interface part "I" answers the question: *How can it be used?*
- Interface is a subset of $2^P \times 2^M \times 2^E$
- Interface can be modified by operations

Compositions

- There are three kinds of compositions among device beans:
  1. Add (+): two device beans are added together without direct interactions.
  2. Multiply (*): Hook up an event from a source component to a method in a target component.
  3. Modifications of the interface (p+, p-, m+, m-, e+, and e-)

Addition (+)

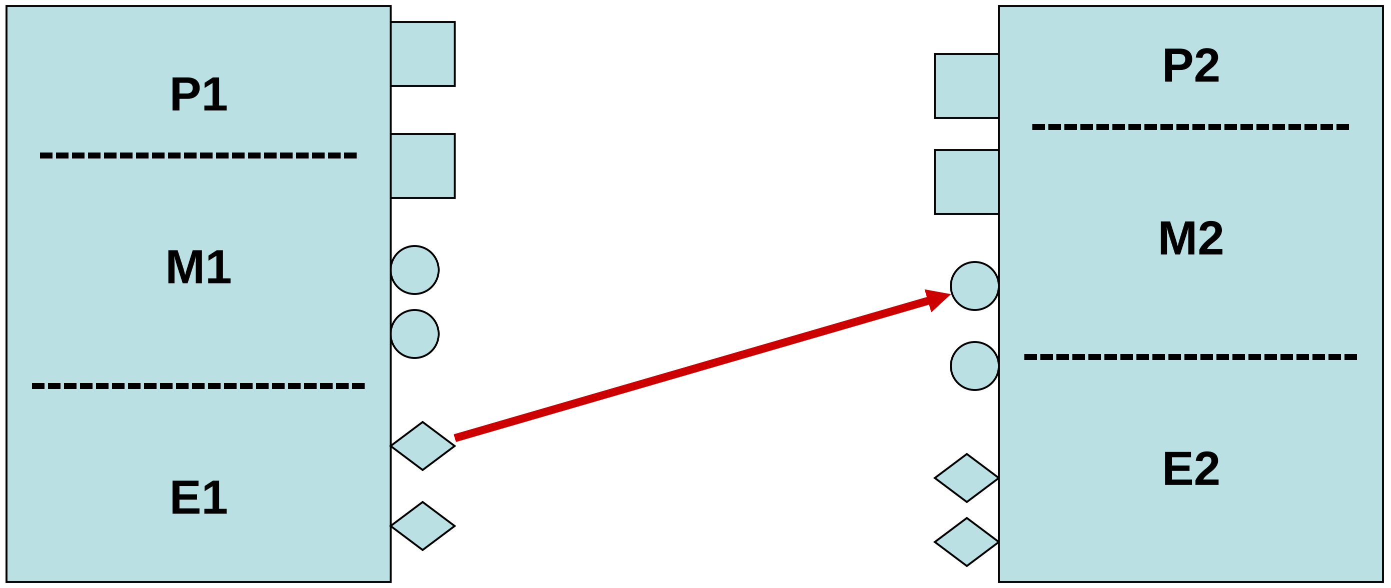
C1 = (P1, M1, E1, I1), C1 = (P2, M2, E2, I2)



C1 + C2

Multiplication (*)

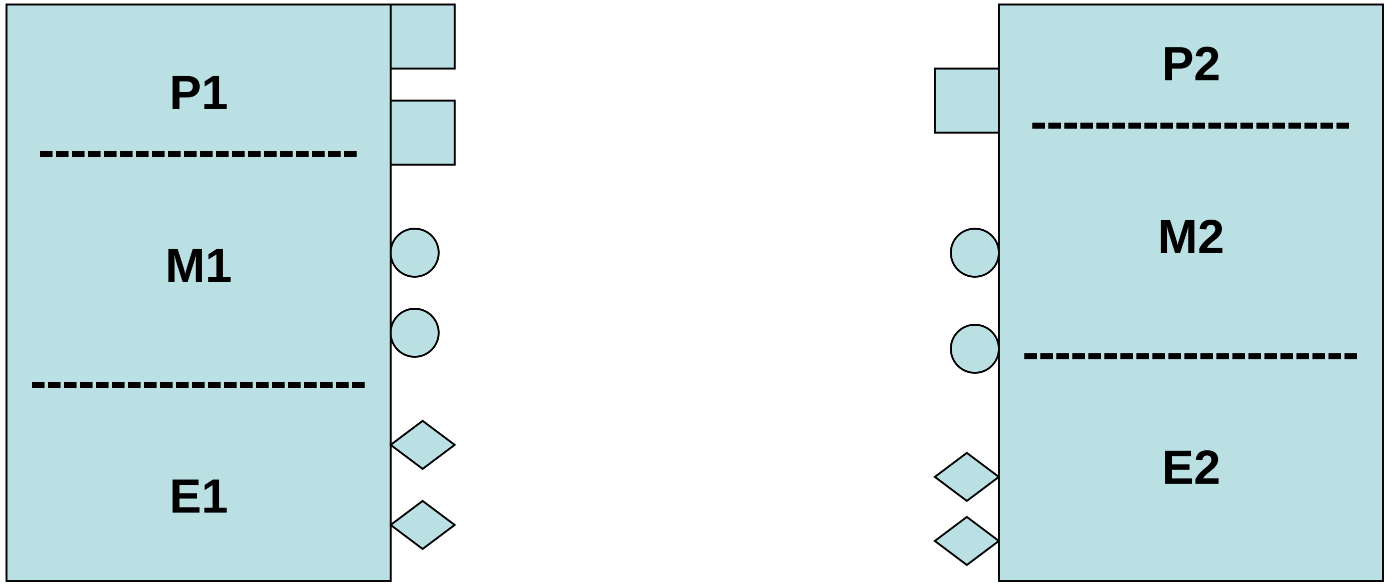
C1 = (P1, M1, E1, I1), C1 = (P2, M2, E2, I2)



C1 * C2

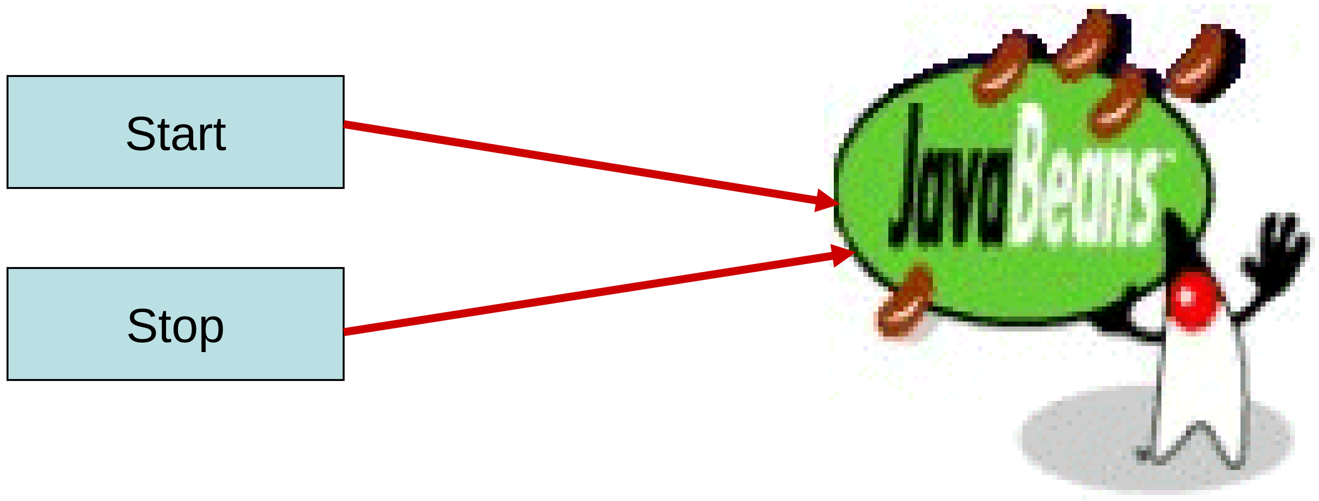
Modification

C1 = (P1, M1, E1, I1), p^- (C1) = (P2, M2, E2, I2)



P^- (C1)

Example



B1 * (start)A + B2 * (stop)A

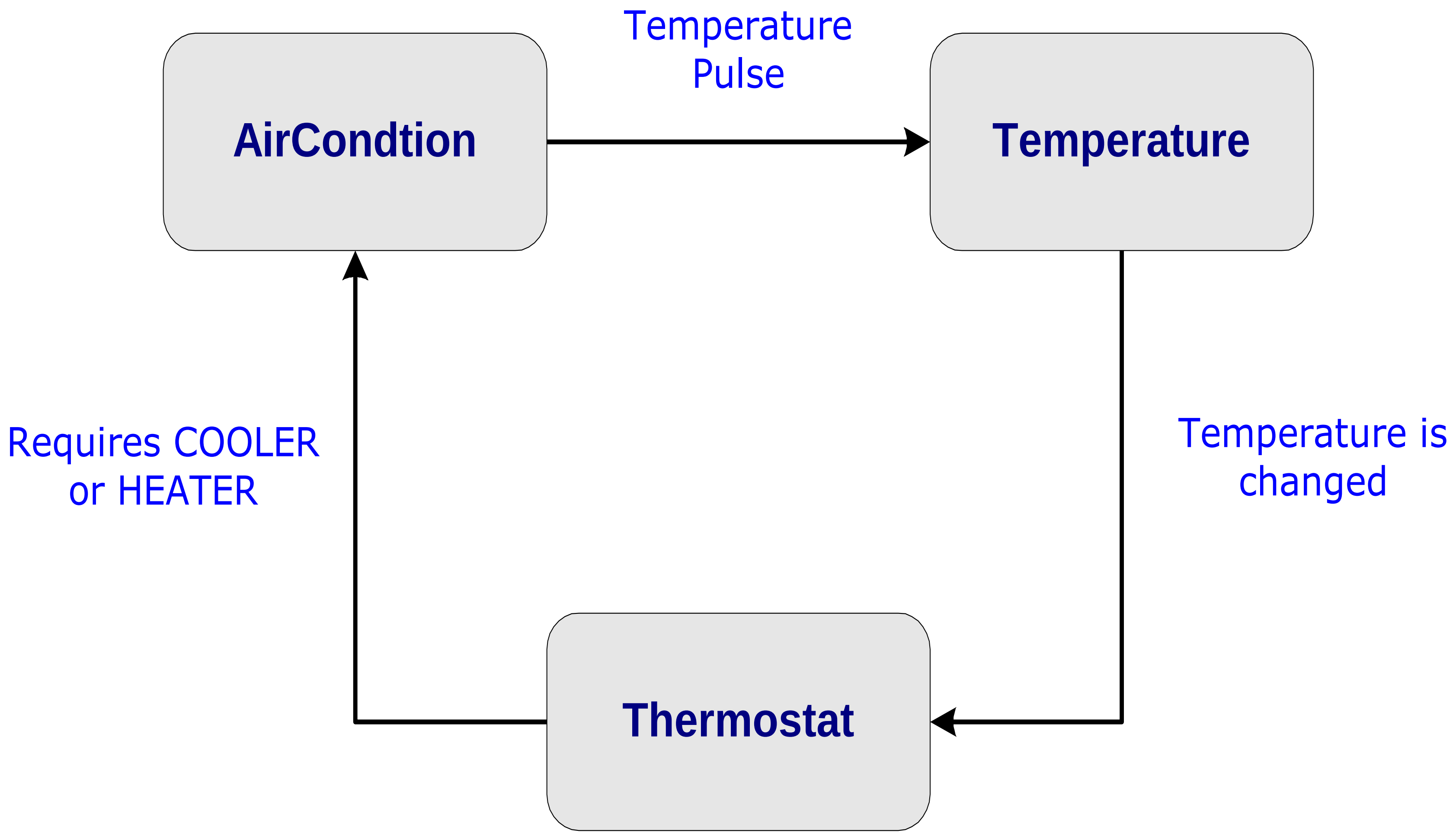
Formal Semantics

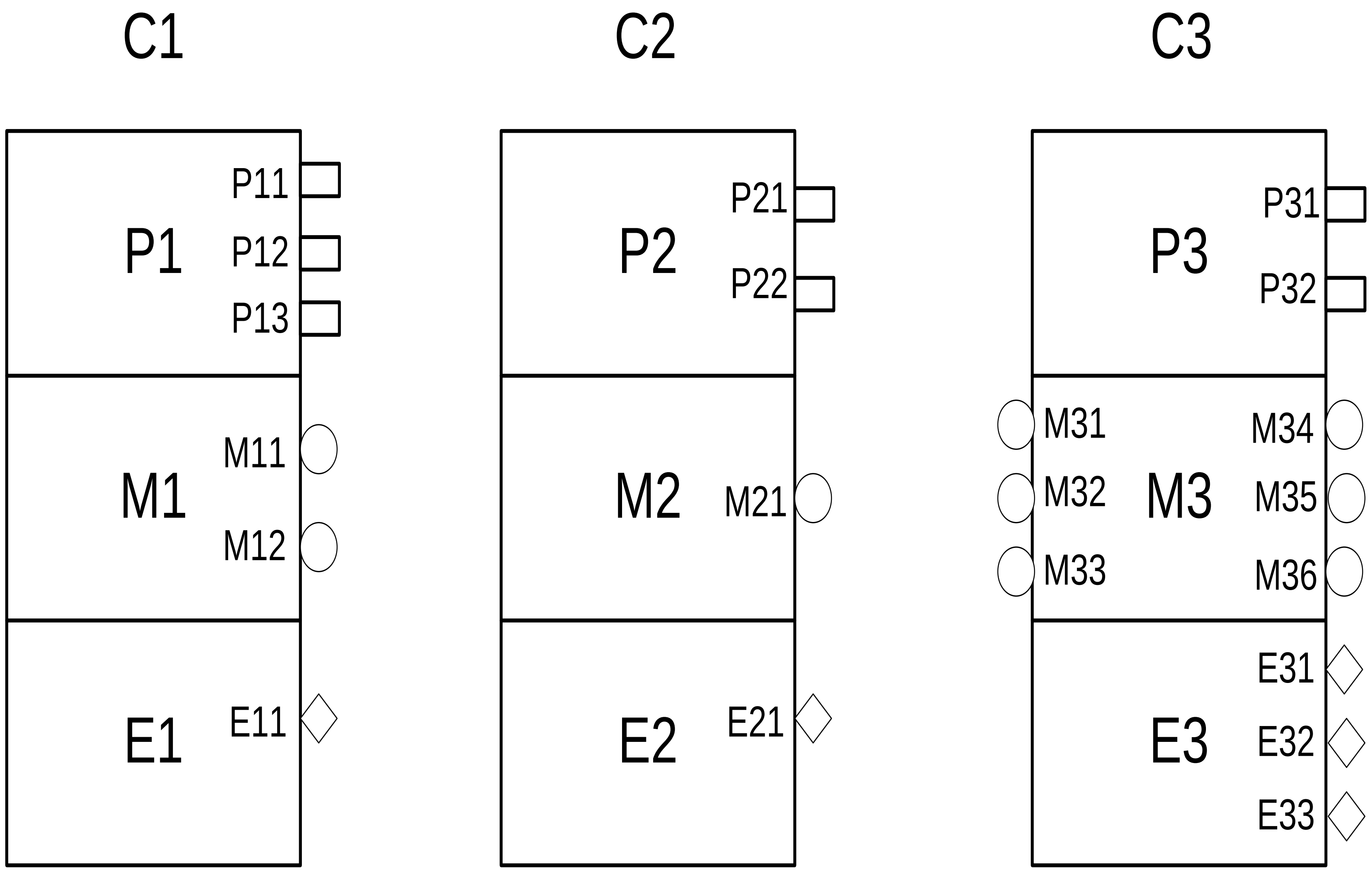
- Each device bean is represented by a *component chart,* which is an extension of statechart, and is similar to objectchart (Coleman et. al.)
- The behavior of a device bean in component communication is described by the *traces* of its component chart.
- The system behavior can be described in terms of component traces

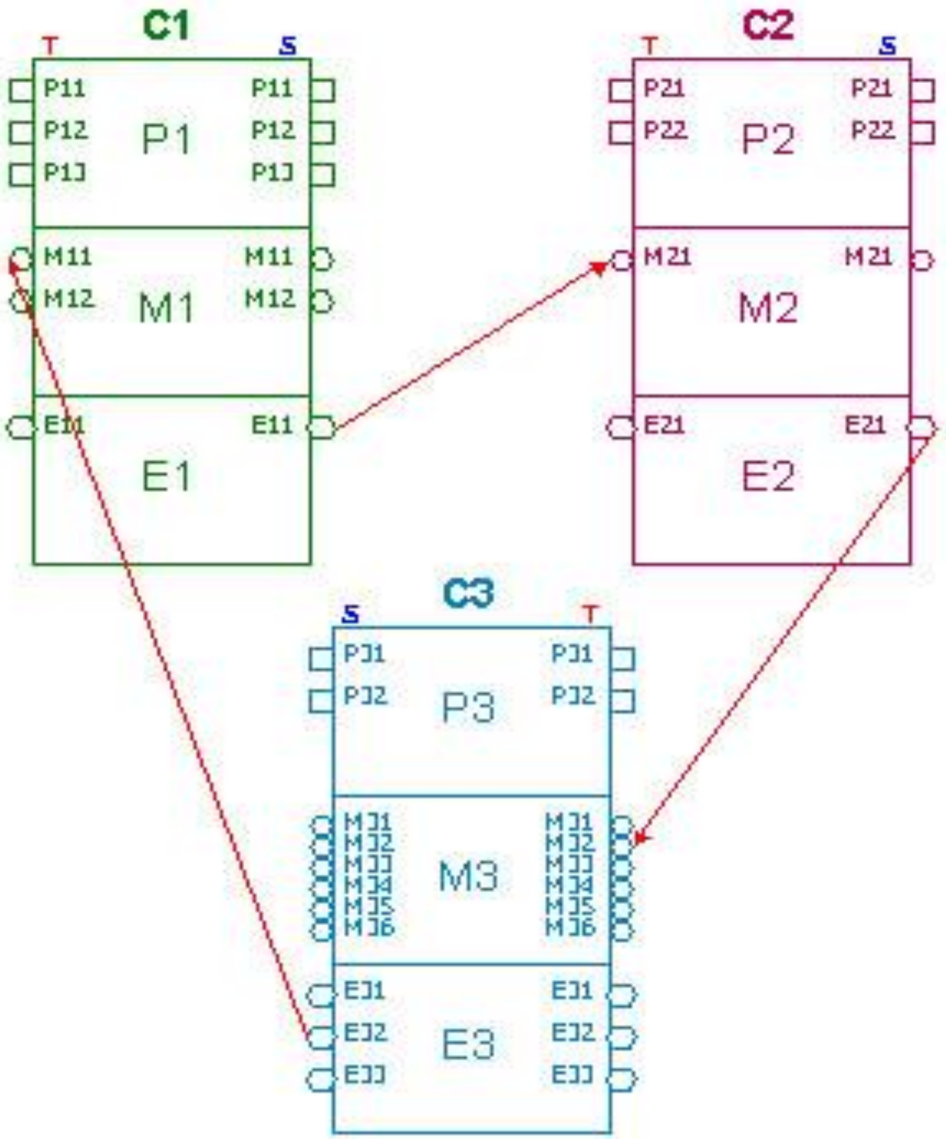
Specifying Contracts

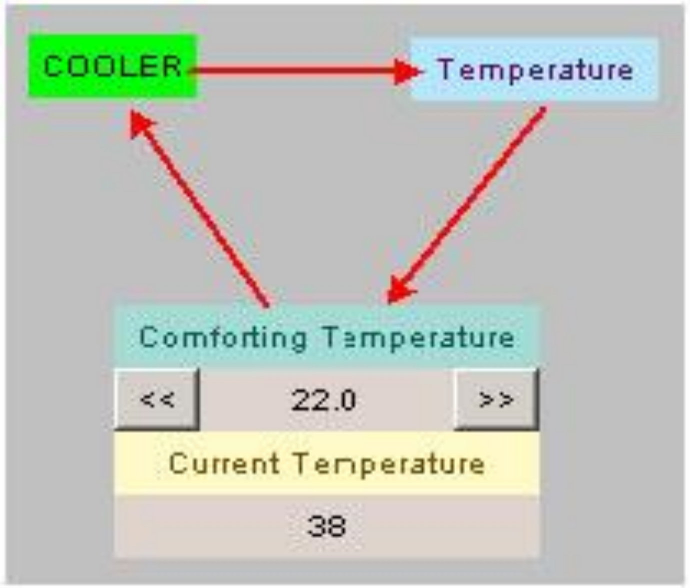
- For each method in an interface of a device bean, a contract is specified using the following:
  - Precondition – a definition of the situations under which the postcondition will apply
  - Postcondition – a description of the effects of that method on its parameters and the information model

Example: The Air Conditioning System




C1
C2
C3
P1
P11
P12
P13
M1
M11
M12
E1
E11
P2
P21
P22
M2
M21
E2
E21
P3
P31
P32
M3
M31
M32
M33
M34
M35
M36
E3
E31
E32
E33

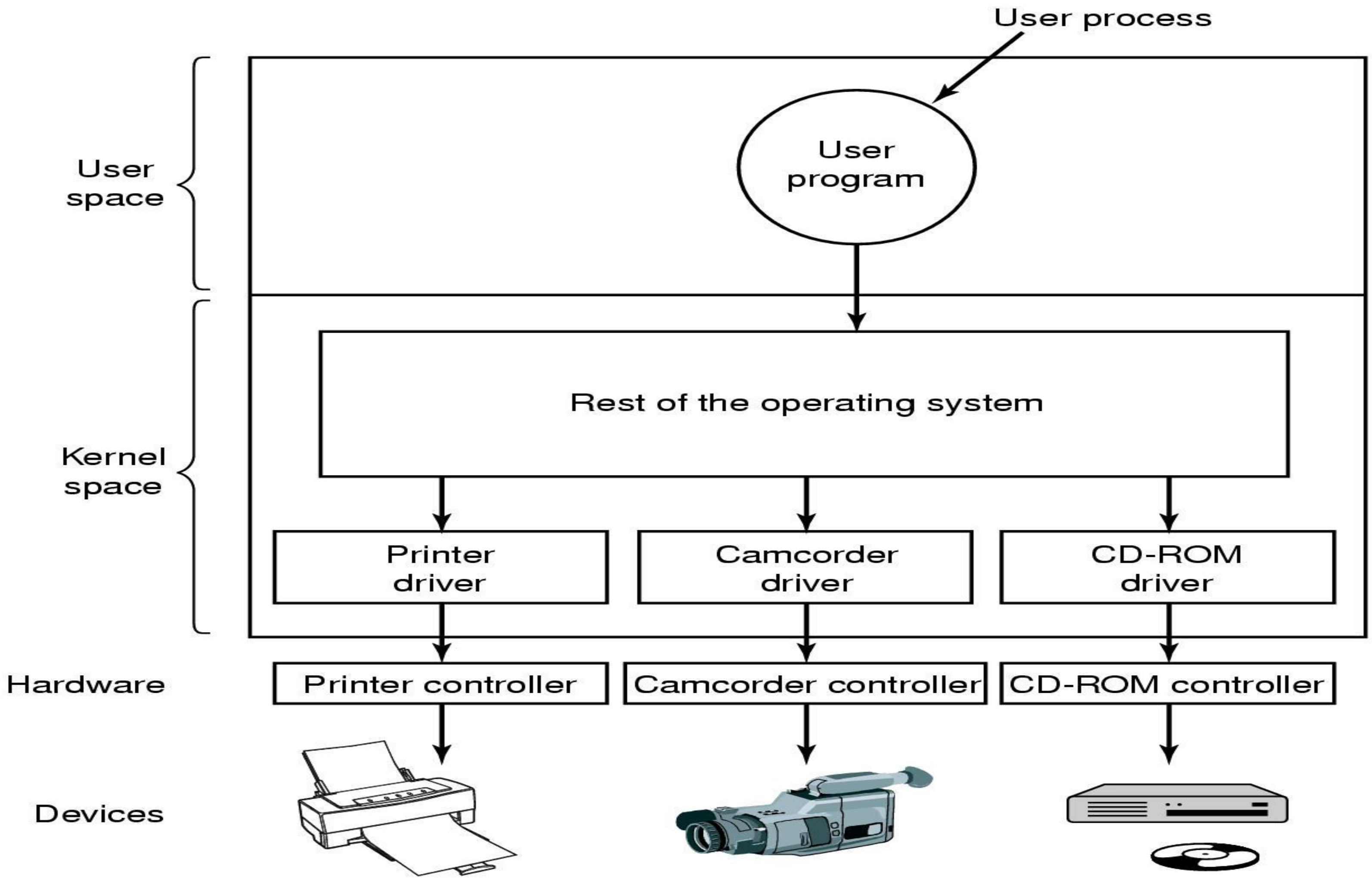

C1
T
S
P11
P12
P13
P1
M11
M12
M1
E11
E1
C2
T
S
P21
P22
P2
M21
M2
E21
E2
C3
S
T
P31
P32
P3
M31
M32
M33
M34
M35
M36
M3
E31
E32
E33
E3


COOLER
Temperature
Comforting Temperature
<<
22.0
>>
Current Temperature
38

Definition of embedded components

- What is a component in embedded systems, informally?
- What is a component in embedded systems, formally?
- For embedded systems, we define embedded components as *device beans* explained below.

Device Drivers



Device beans are basic components

- A typical embedded system can be defined as a collection of devices reading, processing, and controlling physical plant and quantities.
- Example: In an intruder alarm system, motion sensors are devices installed near windows and doors to detect the presence of an intruder. The alarm is a device to generate audible signal. The microprocessor is also a device to collect signals from sensors and to trigger the alarm under certain conditions.

Device beans are reusable

- Especially in a product line, one device is reused in several similar products.
- A sensor controller, for instance, can be reused in a fire alarm system, an intruder alarm system, or a home security system. A timer device, being hardware or software, can be reused in all those embedded systems requiring time service.

Device beans are building blocks

- Embedded systems are usually cost sensitive and require short-time to market.
- Reusable building blocks offer pre-built, thoroughly tested, and ready-to-use components for constructing a new embedded system, thus reducing cost and time.

Device beans are executable

- The object form of device beans are executable.
- We can implement device beans with any programming in principle.
- However, it will be natural to use Java or JavaBeans technology, thus the object forms of device beans are binary components (Java bytecode).
- Rapid prototyping is possible since they could be integrated at design time and executed at run time.

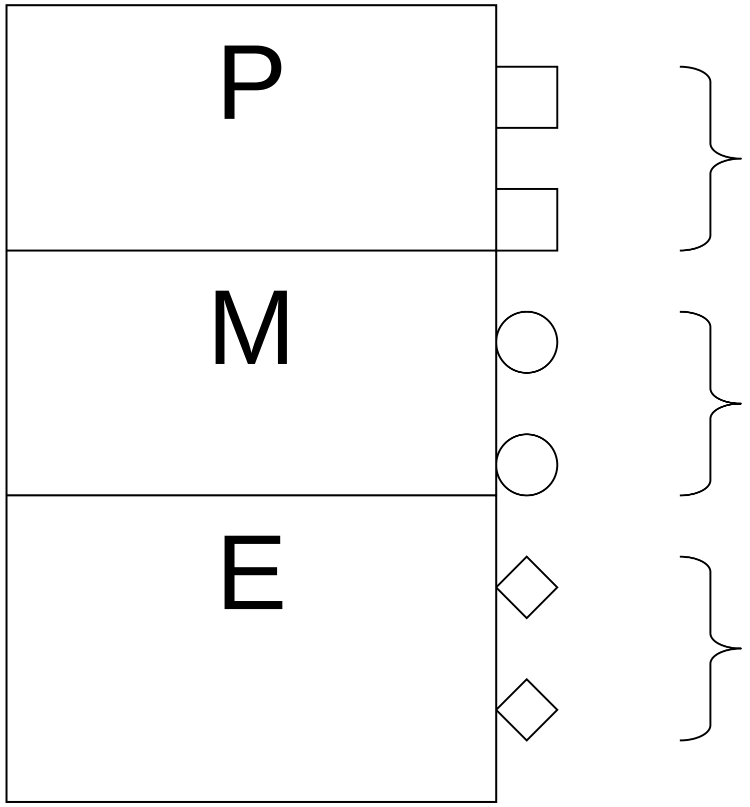
Device beans support visual design

- Visual design provides a convenient and intuitive method to construct a system.
- Since each device bean is implemented in Java, all the visual design tools for Java could be used to design embedded systems. To name just a few of them: JBuilder, Visual Café, Visual Age, Forte for Java, etc. We are building our own IDE to support embedded system design and prototyping.

Device Beans (informal)

- ***Name:*** An identifier presenting the device bean.
- ***Property:*** Properties encapsulate states or attributes of a device bean. Each property has a type.
- ***Method:*** Methods describe behavior and services of a device bean. Each method has a signature : `visibility access return-type Method-name (parameter-list)`
- ***Event:*** Events describe actions this device bean can initiate. Each event has a type.

Graphical Representation



Property Interface

Method Interface

Event Interface

Major differences between device beans and Java beans

- There is a formal definition for device beans specification with formal syntax and formal semantics, while there is no formal definition for Java beans specification yet.
- DBSL (Device Bean Specification Language) defines connectors as operators in component algebra, while Java beans do not have formal connectors defined.
- The component algebra provides several laws for component connections and semantics for each connector, while Java beans do not have formal laws to govern their design-time behavior or run-time behavior.
- After composing two Java beans we got a JAR file which is no longer a component anymore. In device beans, however, two device beans are composed with composition laws and the result is also a component. This makes incremental development with component-based approach possible.

Examples

- A Button device bean
- An animation device bean
- A sensor device bean
- A processor device bean

Summary

- A software component is a piece of self-contained code with well-defined functionality and can be reused as a unit in various contexts.
- A component infrastructure is the basic, underlying framework and facilities for component construction and component management.
- A component infrastructure consists of three models: a component model, a connection model, and a deployment model.
- Device beans are reusable software components for embedded systems.