Components vs. Objects

Luigia Petre

Turku Centre for Computer Science

&

Abo Akademi University,

FIN-20520 Turku, Finland

(*Presented at Nordic Workshop on Programming Theory, Bergen, Norway, Oct 11-13, 2000*)

Software Engineering Abstractions

- Component-based SE = constructing new systems from already existing, service-providing components

- Object-based SE = constructing a new system in terms of interacting, distinct units of information and services called objects

Motivation

Both components and objects

- have encapsulation properties
- are accessed via well-defined interfaces
- are considered to improve the reuse of software
- are considered to alleviate the software evolution phase
- are thought of being natural abstractions of real-world entities
- a real-world entity can be modelled / implemented using either notion

Our Aim

...is to answer to:

(*a*) What is the essential difference between components and objects?

(*b*) (Why) do we need them both?

Components (1)

Software components = units of composition with contractually specified interfaces and context dependencies only

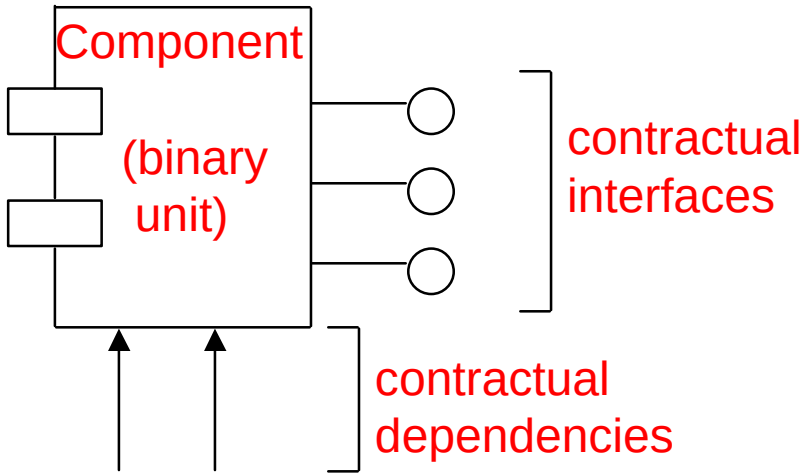
A SW component

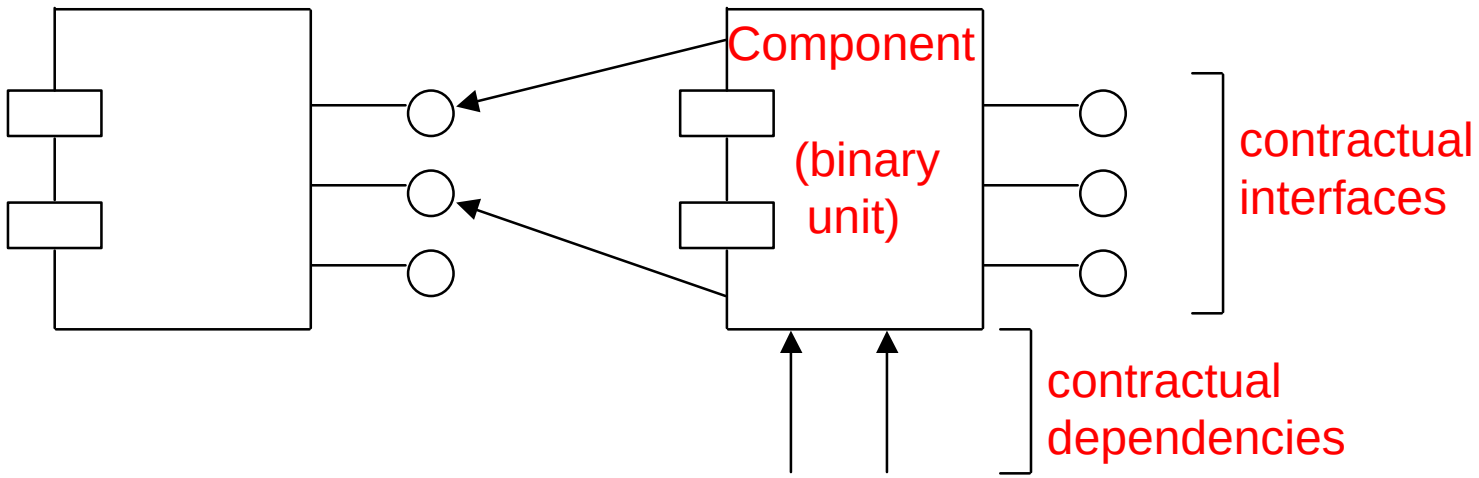
- can be deployed independently
- is subject to composition by third parties

Components (2)

=> possibly 3 parties involved:

- one specifies the component,
- one implements the specification of a component,
- one deploys / use the component




Component
(binary
unit)
contractual
interfaces
contractual
dependencies

Components (3)

A system built up from components

- is more robust
- is more flexible (alleviates evolution)
- has a shorter development time / process

The foremost advantage: reuse of software

Objects (1)

Object - abstraction from a real-world entity, with associated items of information and a set of specific operations

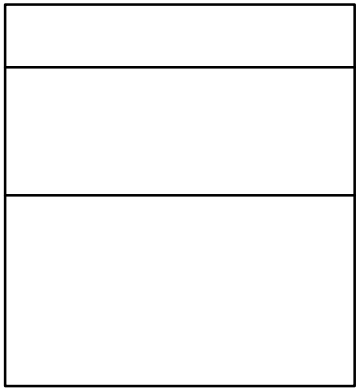
It has

- a unique and invariant identifier
- a class to which it belongs
- a state that has a certain value

Classes (1)

Class - abstract data type with a set of properties (attributes and operations) common to its objects

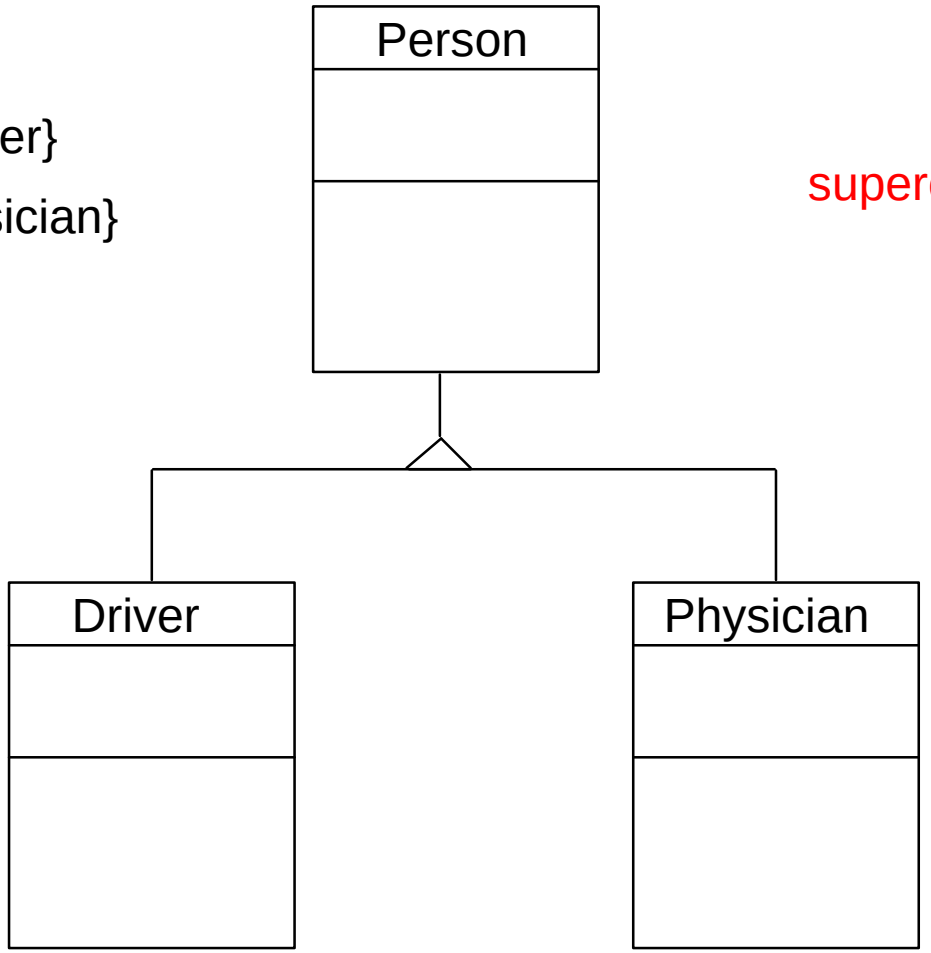
- has the means of creating objects with these properties



Classes (2)

?Person} ? ??Driver}

?Person} ? ??Physician}



superclass / base class

inheritance

subclasses /
derived classes

Objects (2)

Objects interaction -- using each others operations

Objects visibility -- the state and the implementation of the operations is hidden

A system built up from objects

- is modular (*designing classes rather than the whole system*)
- is reusable (*inheritance*)
- alleviates evolution (*objects are fundamental and stable, implementation is hidden*)

The conceptual difference: their role

Objects

- describe / implement real-world entities (and their hierarchies)
- mathematical modelling approach to software
- partition the state space

Components

- describe / implement services of real-world entities
- engineering approach to software
- partition the service space

Example: Mail Delivery System (MDS)

Services:

- input mail
- send mail
  - load mail into the transportation means
  - transport mail to destination
- confirm delivery

MDS Component View

We need a component (PostCar) that is able

- to move
- to have a certain loading capacity

We need a component (Driver) that is able

- to drive for x hours in a row
- to drive certain types of cars
- to receive a salary of no more than y

MDS Object View

We need a class PostCar, with

- a move operation
- a capacity attribute

We need a class Driver, with attributes in some ranges

Components vs. Objects = Service vs. Identity

Using a service from a component:

specify the required service and use any component providing an implementation of that service

Using a service from an object:

specify what object is used and use the service that particular object (with the particular state) can provide

Using Components and Objects

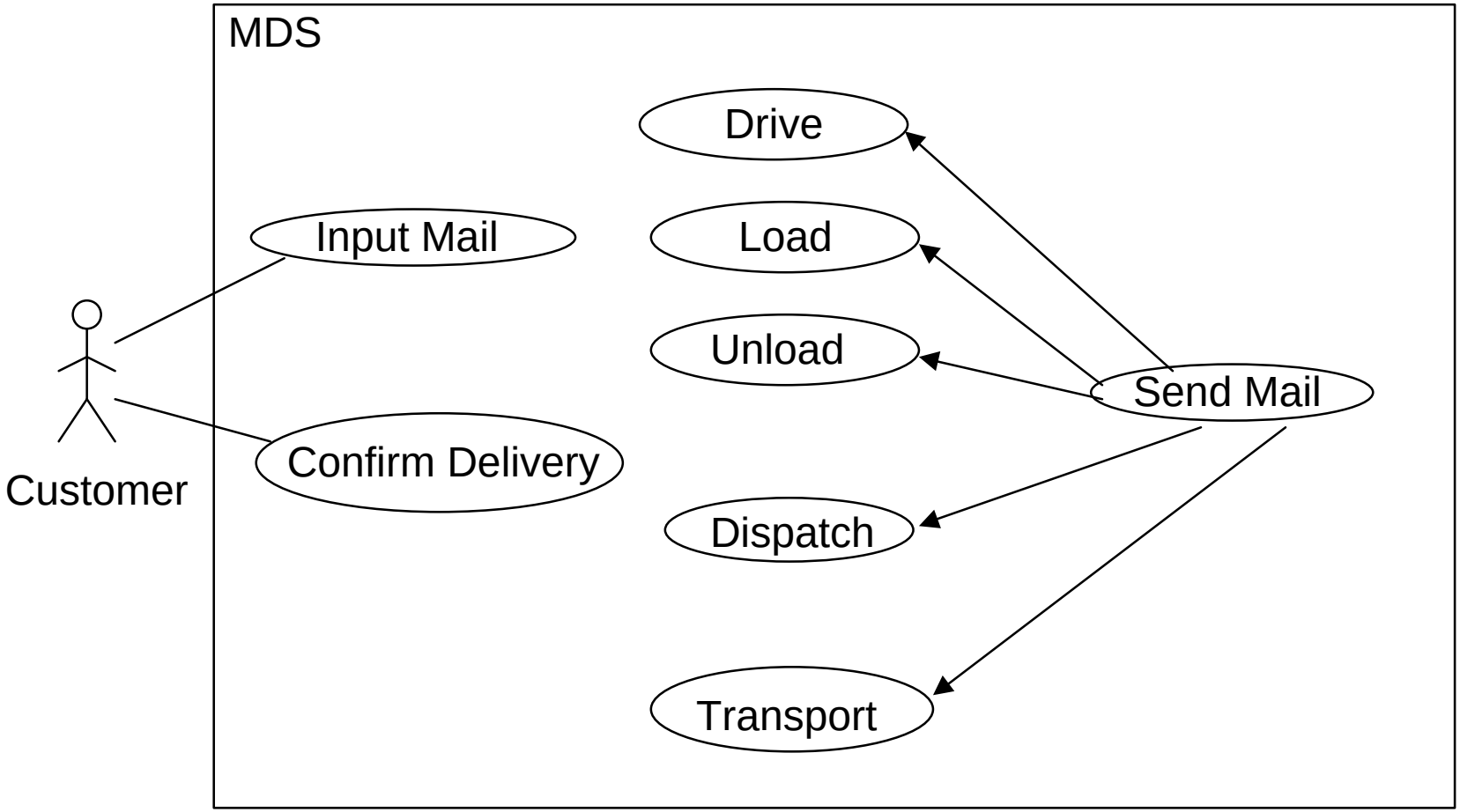
Components -- service-oriented => they describe best the functionality of a system

Objects -- identity-oriented => they describe best the problem domain of a system

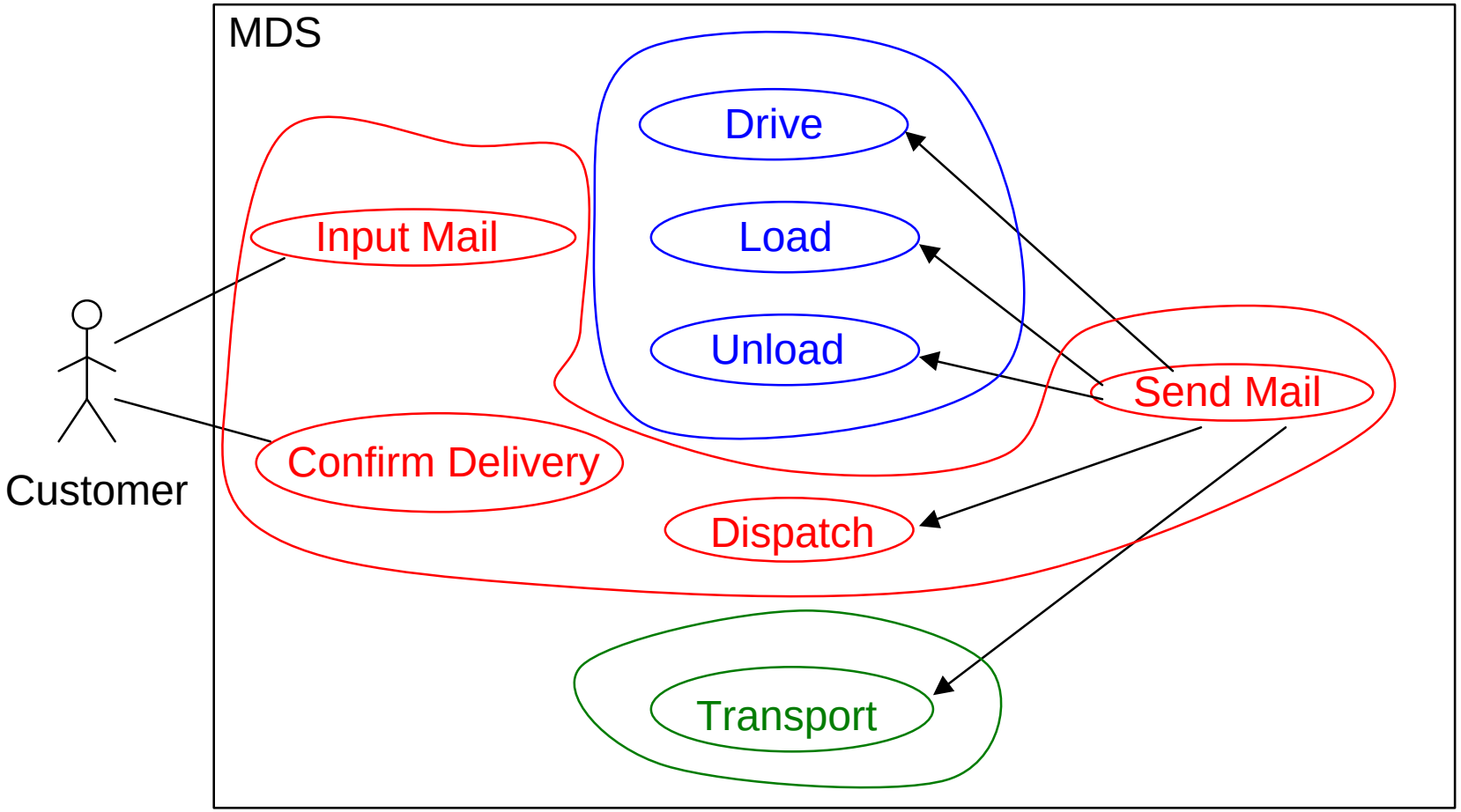
Consequently, we should

- start the software development with components
- develop each component in terms of objects

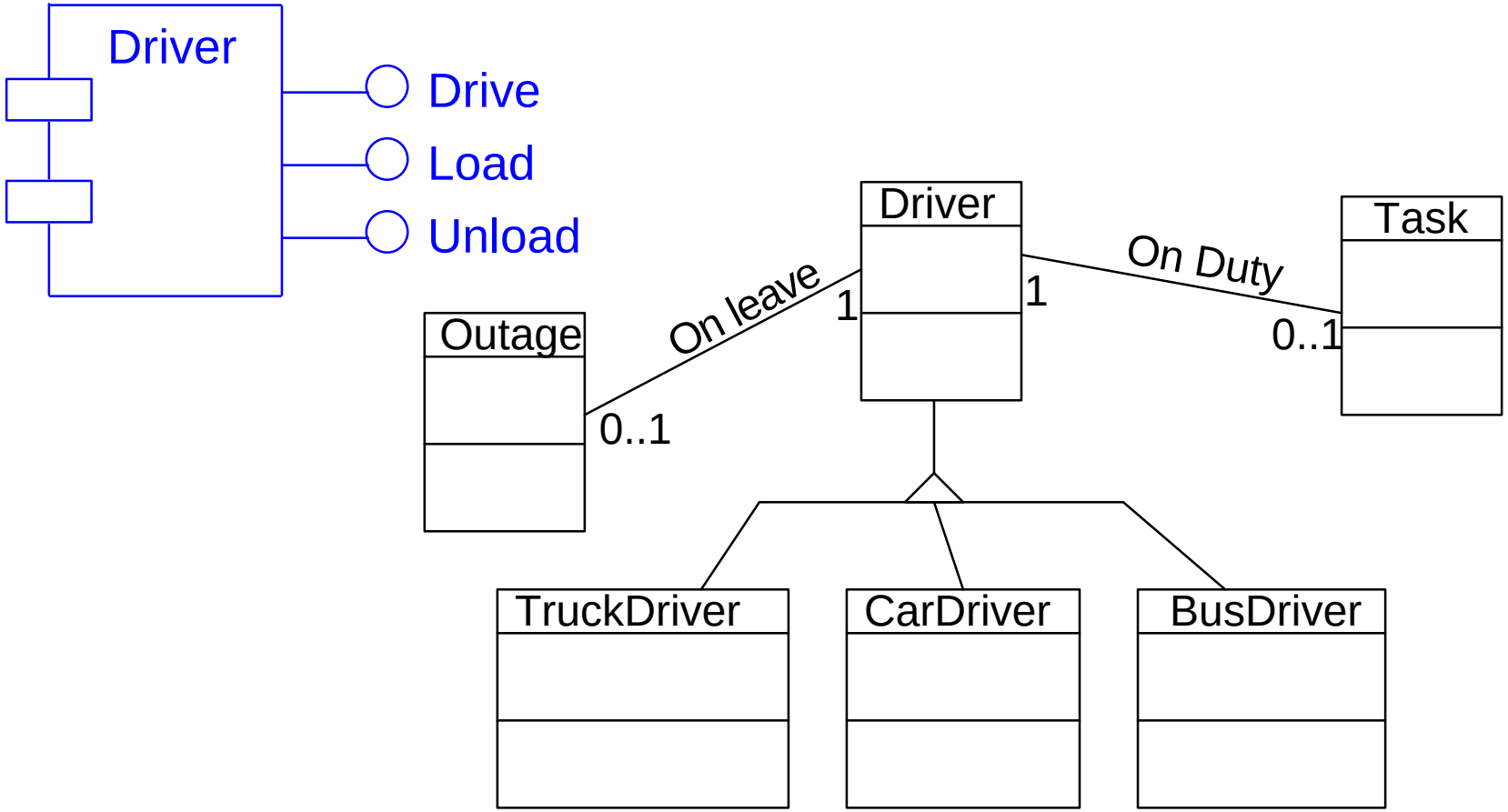
UML Diagrams Outline (1)



UML Diagrams Outline (2)



UML Diagrams Outline (3)



Conclusions

- Components -- service-oriented => best as *functional* abstractions
- Objects -- identity-oriented => best as *problem domain* abstractions
- Components and objects -- best as *software construction* abstractions