CBSE

Component Based Software Engineering

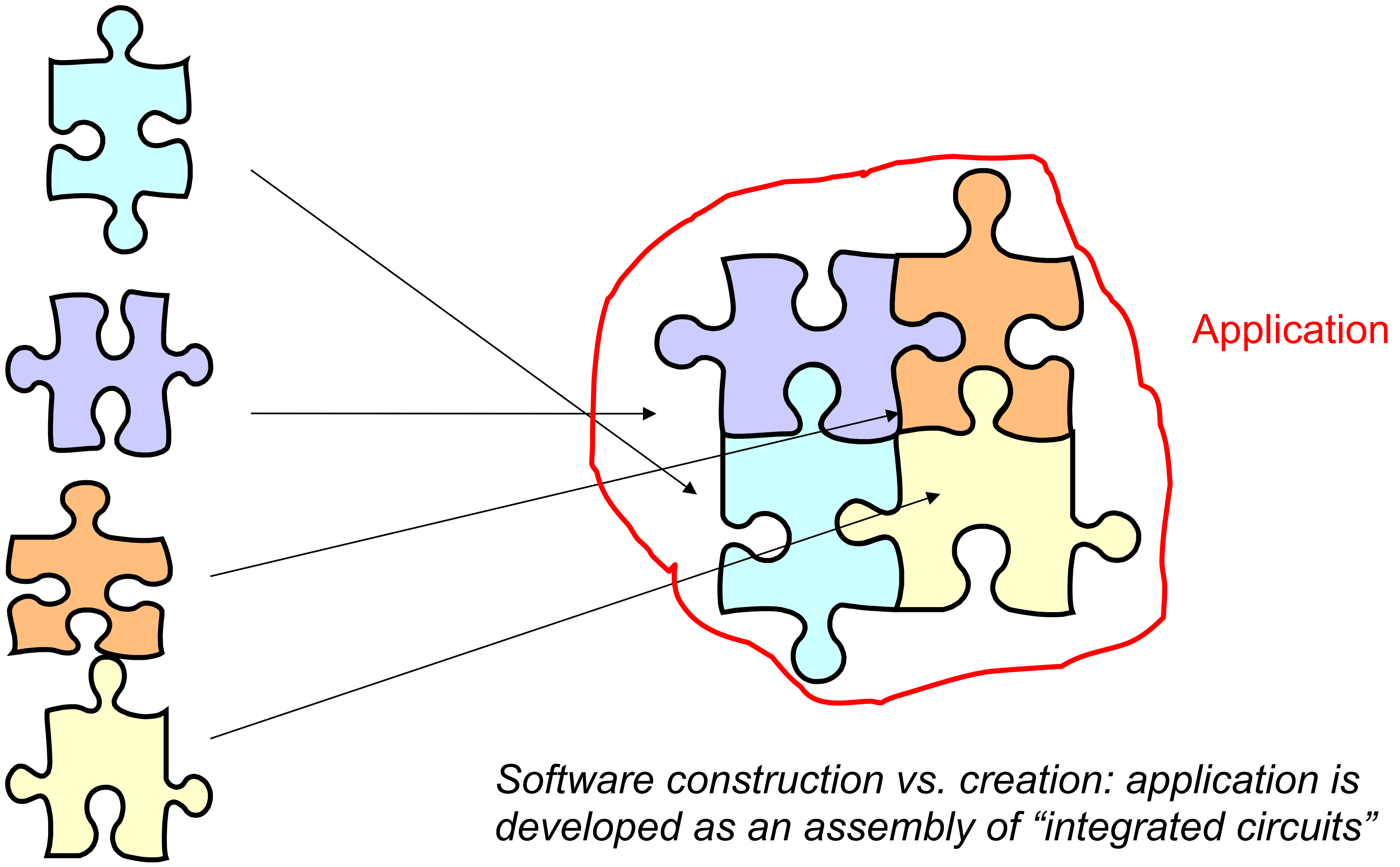
Component based development

"Systems should be assembled from existing components"

- Idea dates since 1968: Douglas McIllroy: "Mass produced software components"

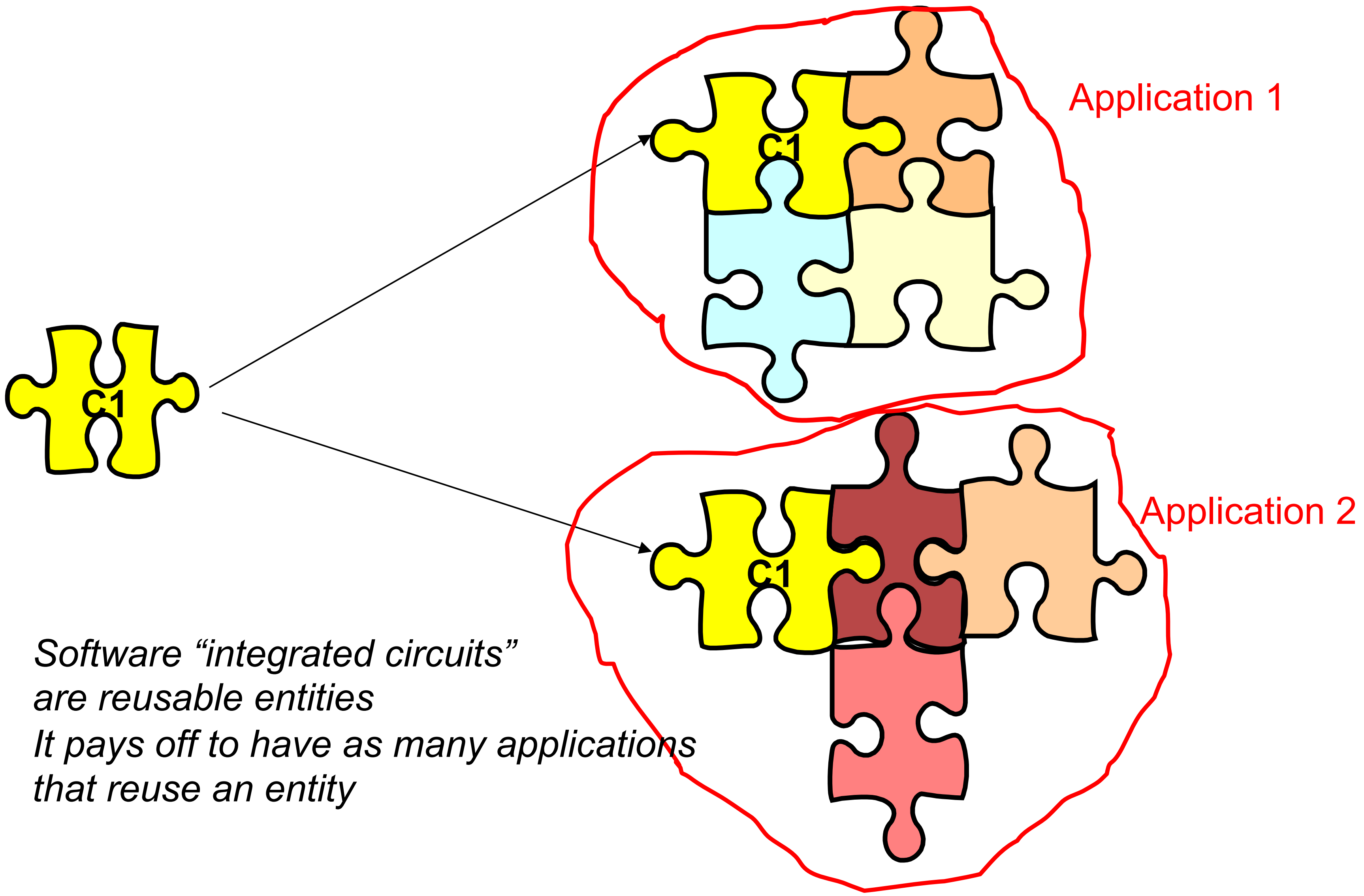
- Component-based software engineering (CBSE) is an approach to software development that relies on software reuse – reusing *artifacts* (*software parts*)
- Advantages of CBSE:
  - **Reuse**: Development of system = assembly of component
  - **Flexibilit**y: Maintenance, upgrading=customization, replacement of components, extensibility by adding components. This may even happen at run-time with proper infrastructure support !

Advantage 1: Software construction



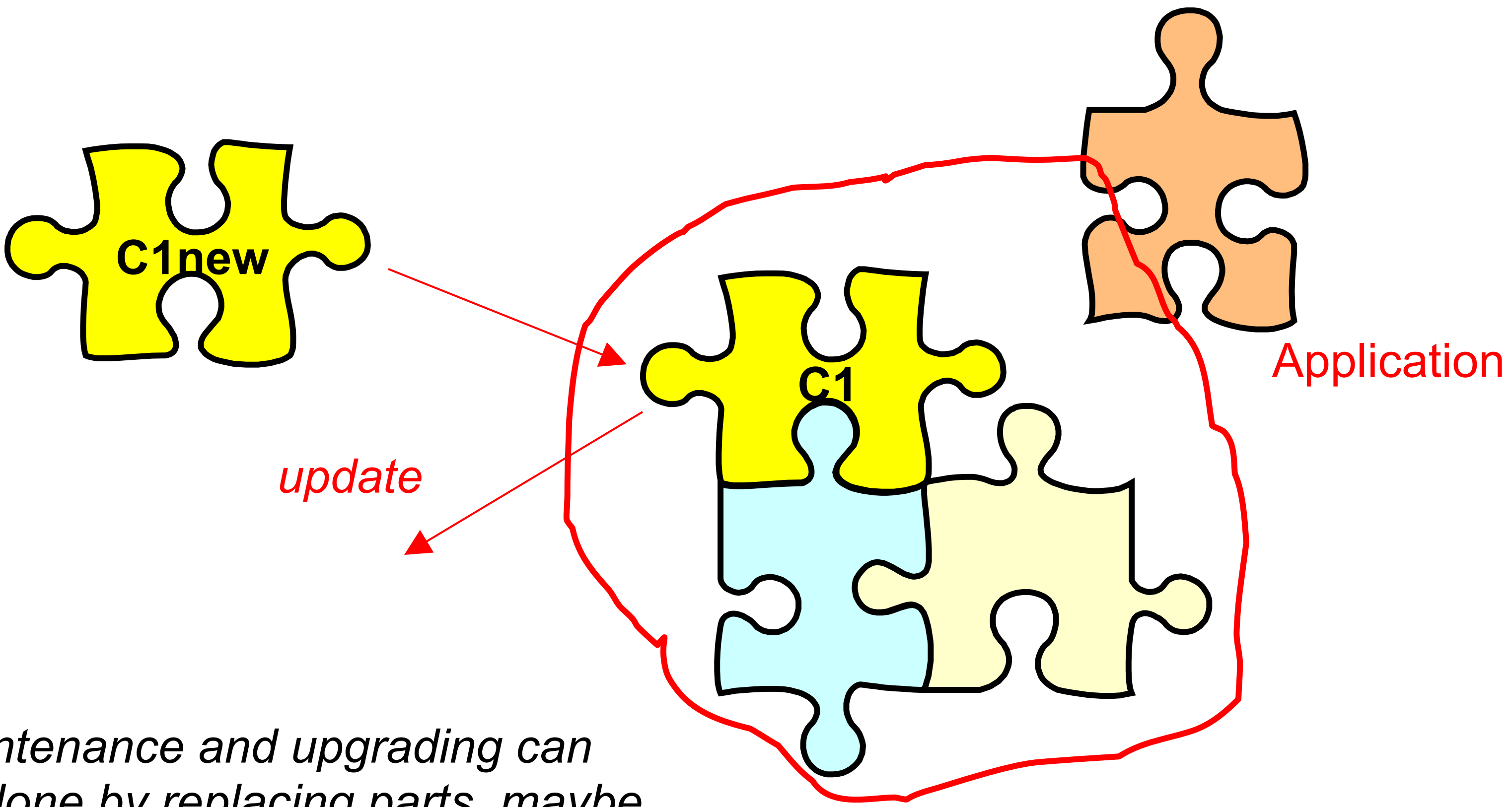
Software construction vs. creation: application is developed as an assembly of "integrated circuits"

Advantage 2: Reuse



Software "integrated circuits"
are reusable entities
It pays off to have as many applications
that reuse an entity

Advantage 3: Maintenance & Evolution



Maintenance and upgrading can be done by replacing parts, maybe even at runtime

Why CBSE Emerged

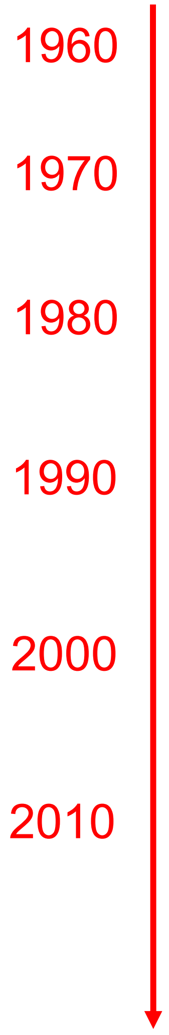
- It is emerged from the failure of object-oriented development to support effective reuse.

  Single object classes are too detailed and specific.

  Components are more abstract than object classes and can be considered to be standalone service providers

What are the “Entities” to compose ?

- Functions
- Modules
- Objects
- Components
- Services
- …



1968: Douglas McIlroy: “*Mass Produced Software Components*”

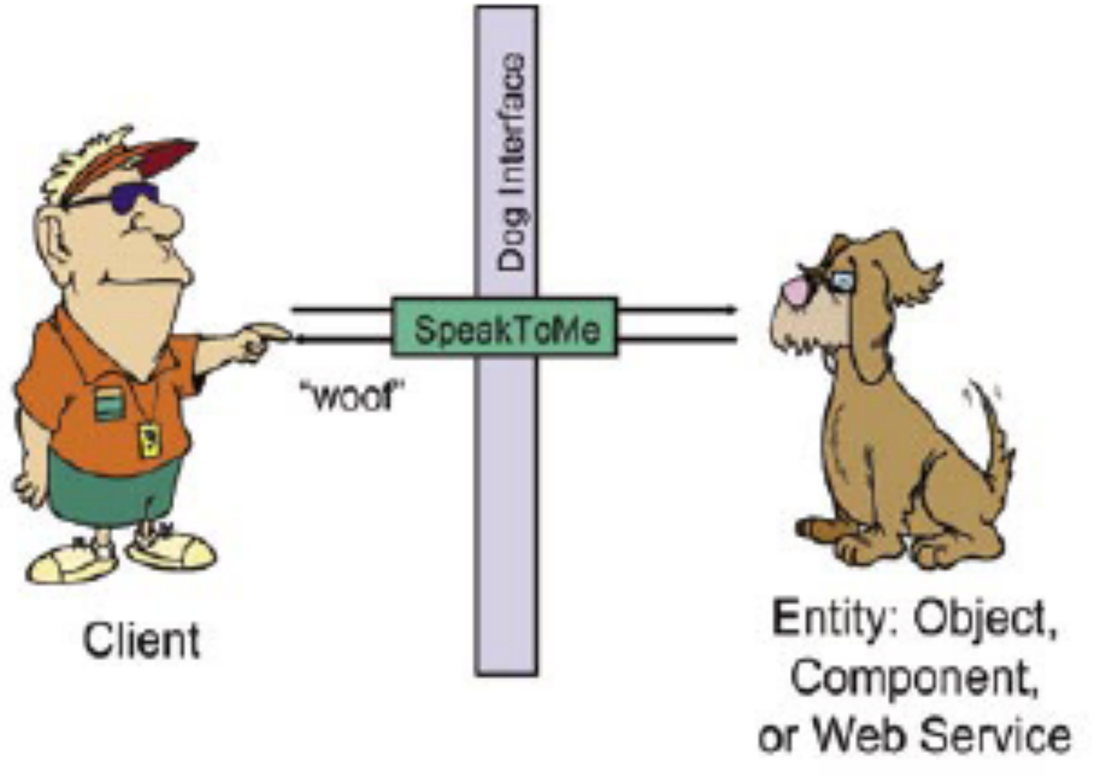
1998: Clemens Szyperski: “*Component Software – Beyond Object Oriented Programming*”

Principles for reuse by composition

- Key requirements for Black-Box reuse:
  - **Abstraction**: an “Entity” is known by its “interface”
  - **Encapsulation**: the “insides” of an “Entity” are not exposed to the outside

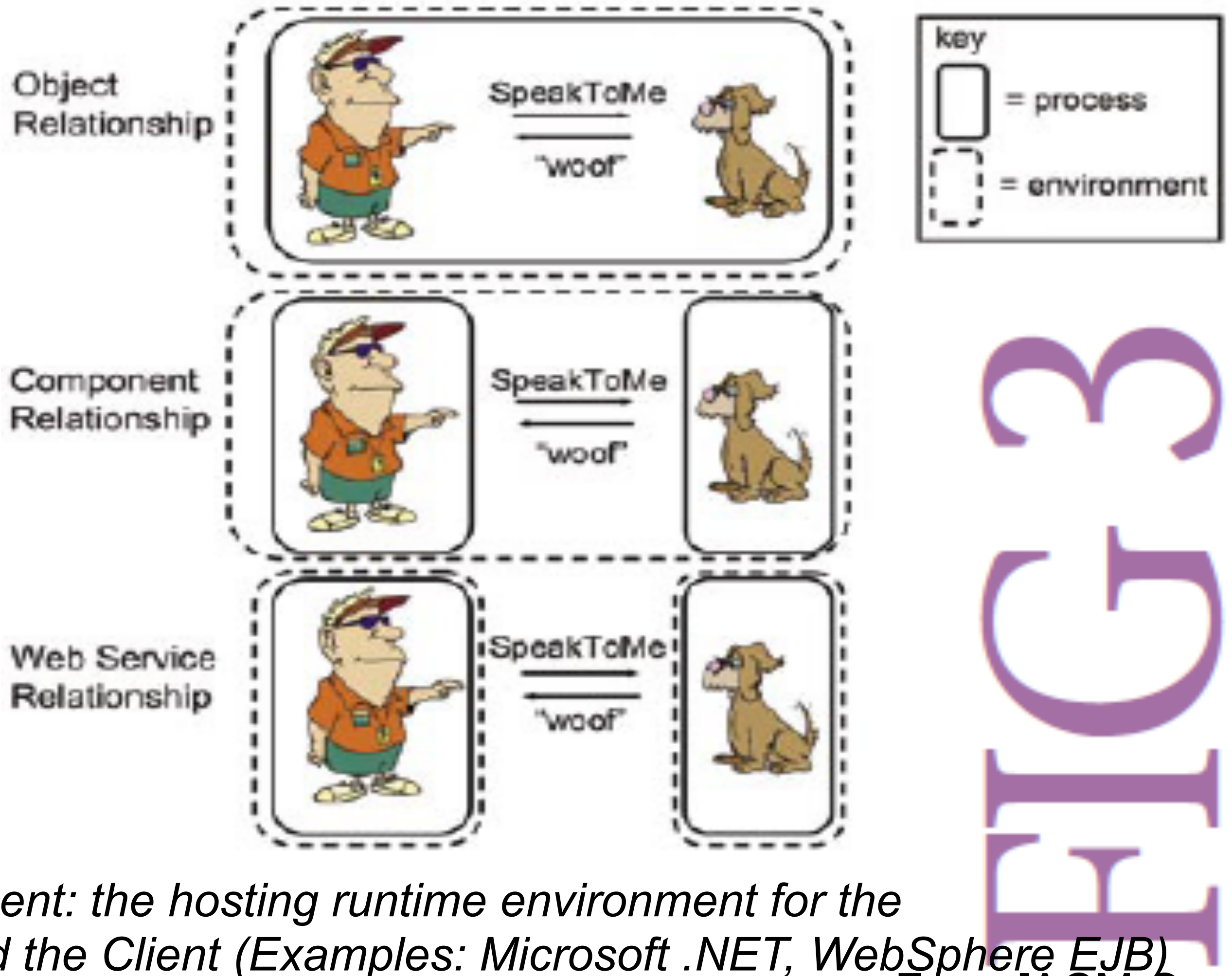
Commonalities of Reusable Entities

- All are blobs of code that can do something
- All have interfaces that describe what they can do.
- All live in a process somewhere.
- All live to do the bidding of a client.
- All support the concept of a client making requests by "invoking a method."



From [ACM Queue]

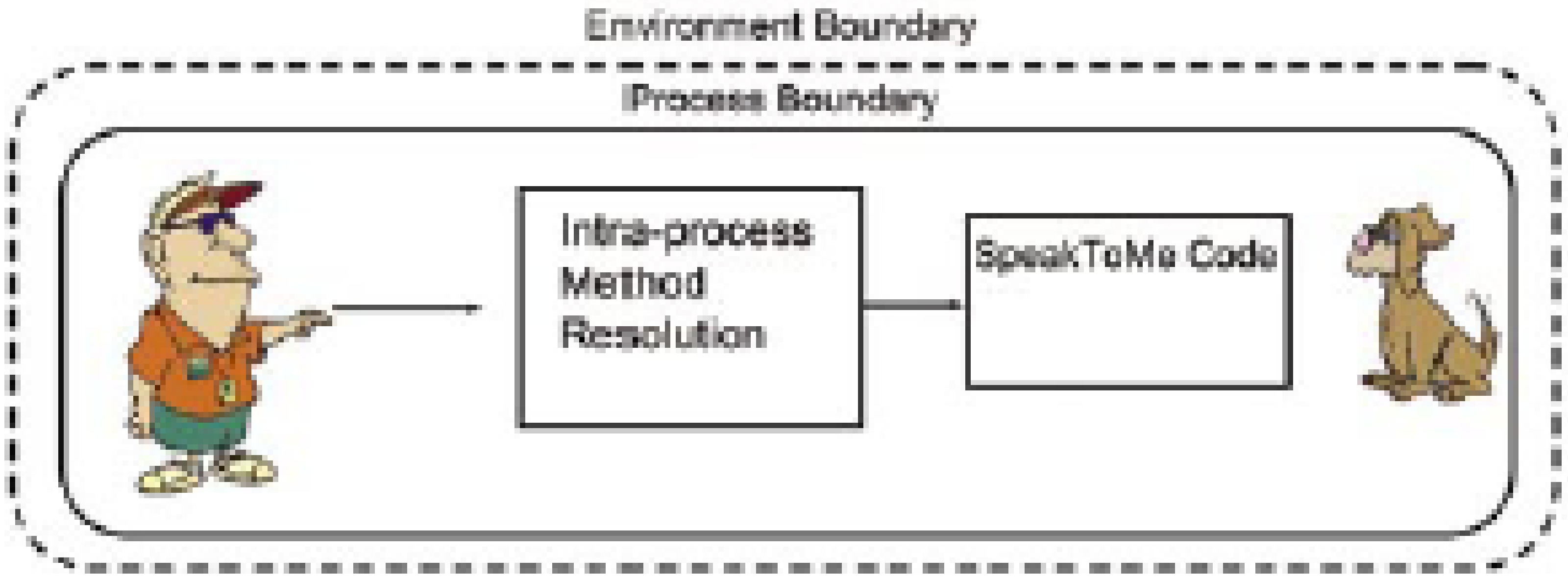
Reusable Entities by Location and Environment



Environment: the hosting runtime environment for the Entity and the Client (Examples: Microsoft .NET, WebSphere EJB)

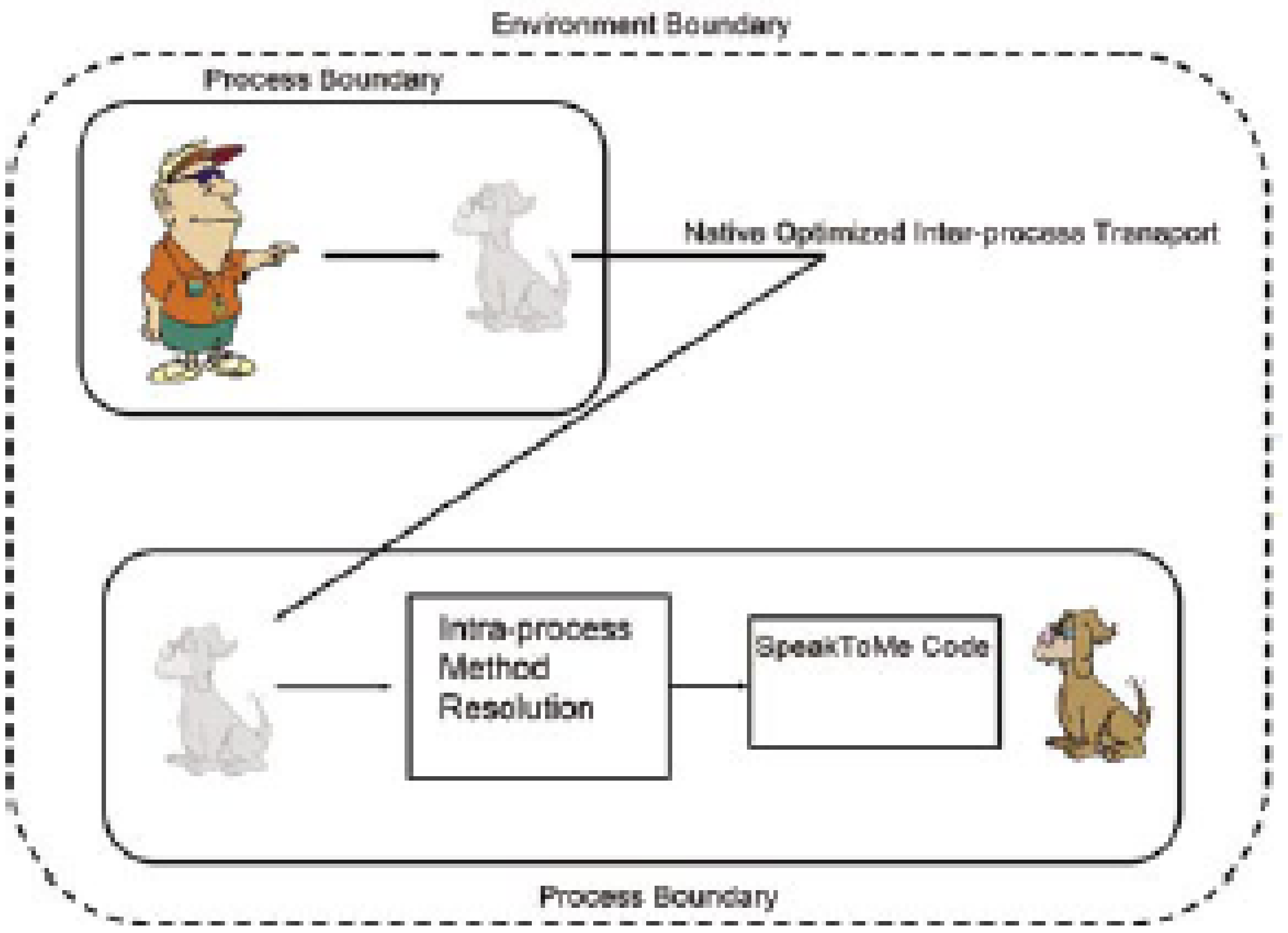
From [ACM Queue]

Object Method Resolution



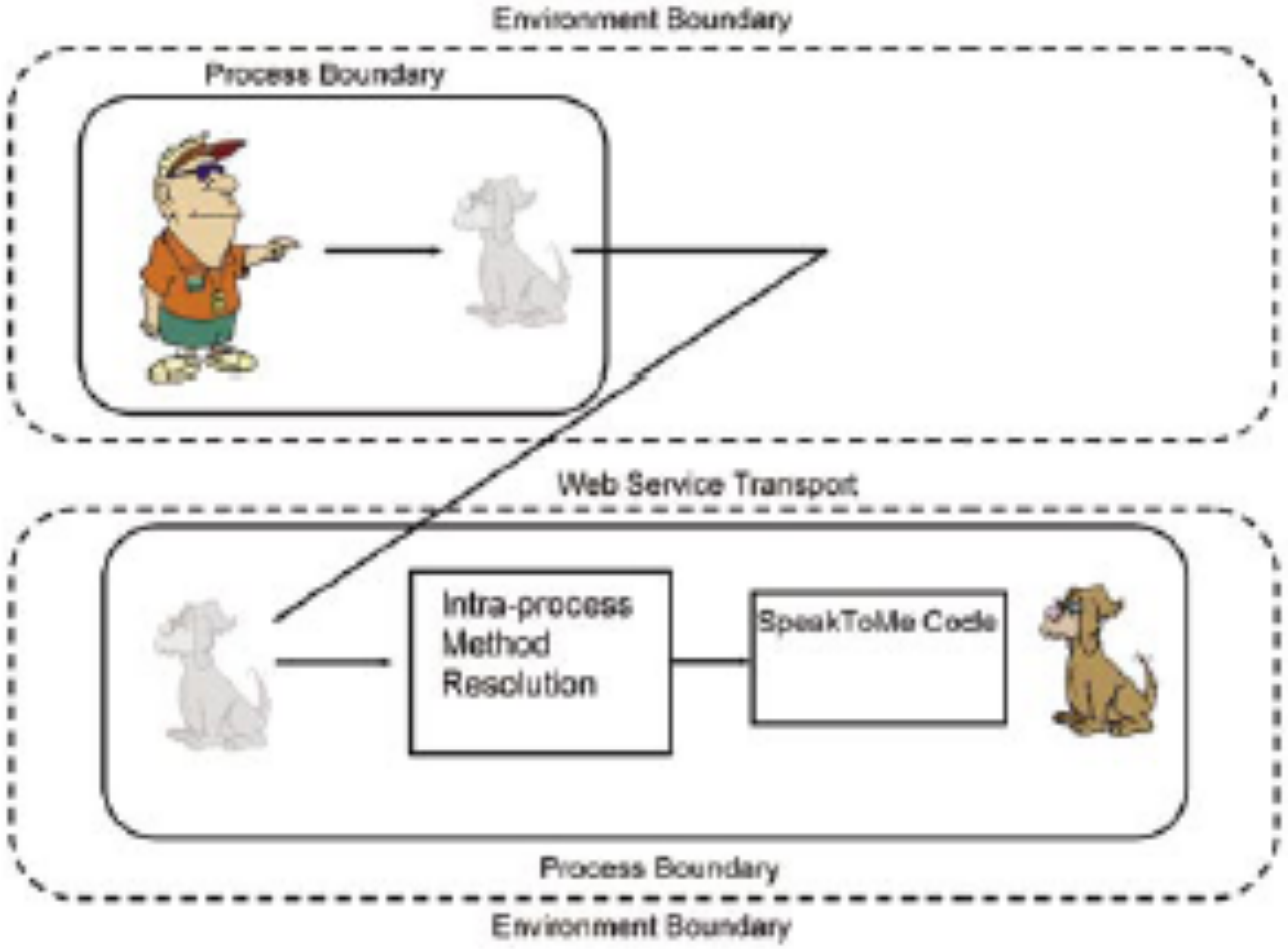
From [ACM Queue]

Component Method Resolution


Environment Boundary
Process Boundary
Native Optimized Inter-process Transport
Intra-process Method Resolution
SpeakToMe Code
Process Boundary

From [ACM Queue]

Web Service Method Resolution



From [ACM Queue]

Hiding component internals

1. Black box: only specification is known
2. Glass box: internals may be inspected, but not changed
3. Grey box: part of the internals may be inspected, limited modification is allowed
4. While box: component is open to inspection and modification

Objects-Components-Services

Entities for Reuse and Composition		
•Abstraction •Encapsulation		
Objects	**Components**	**Services**
•Location: same process •Inheritance •Polymorphism	•Location: different processes, same environment •Usually some runtime infrastructure needed •No state •No shared variables	•Location: different environments •More emphasis on interface/contract/service agreement •Mechanisms for dynamic discovery •Dynamically composable

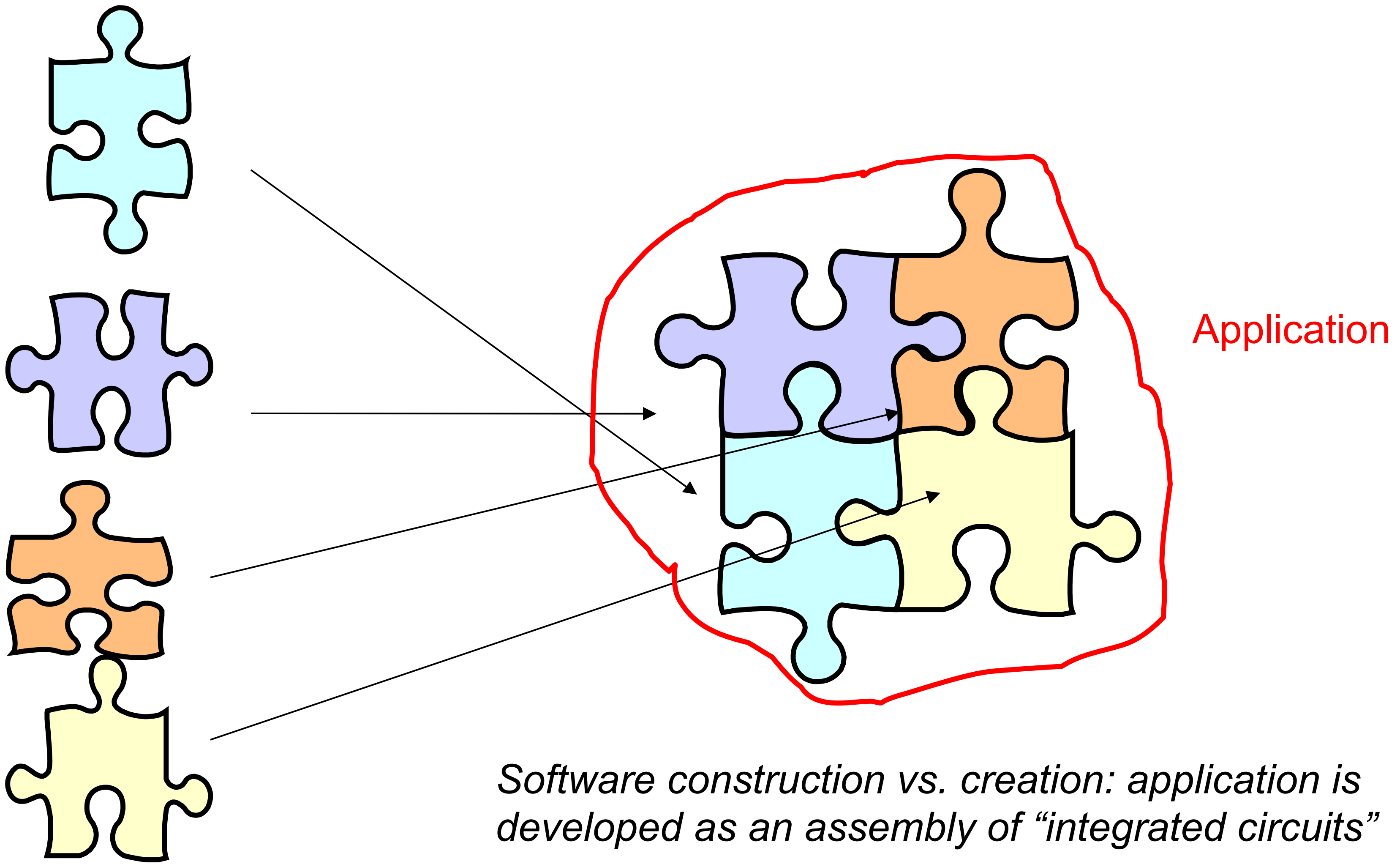
Reusable Entities made more usable and more composable

- Issues:
  - Interface description – what should contain a complete description ?
  - Composition – how are components glued together ? (do I have to write much glue code ?)
  - Discovery – where and how to find the component/service you need ?
  - Dynamic aspects – when to do discovery/selection/composition
  - Less stress on binary implementation – crossing platform/model boundaries

CBSE reuse

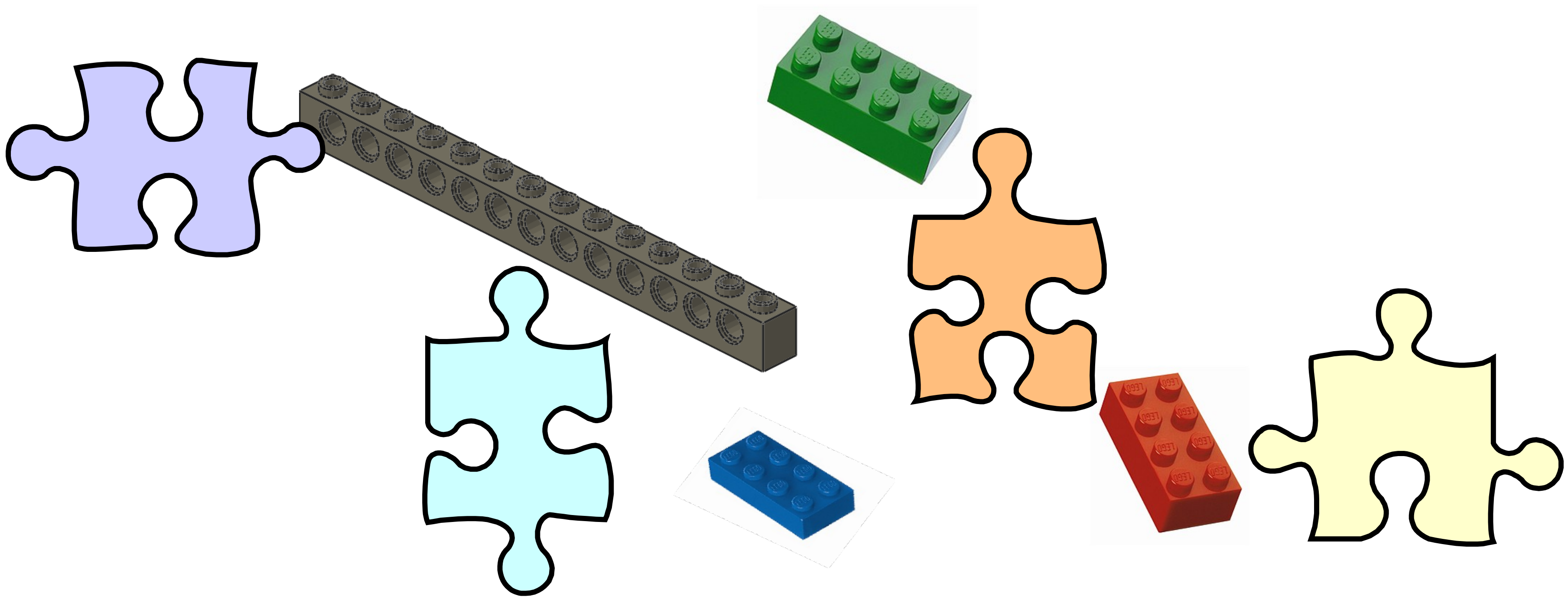
- **Component Based Software Engineering (CBSE) = reuse of:**
  - **Parts (components)**
  - **Infrastructure**

Component based software construction – the ideal case



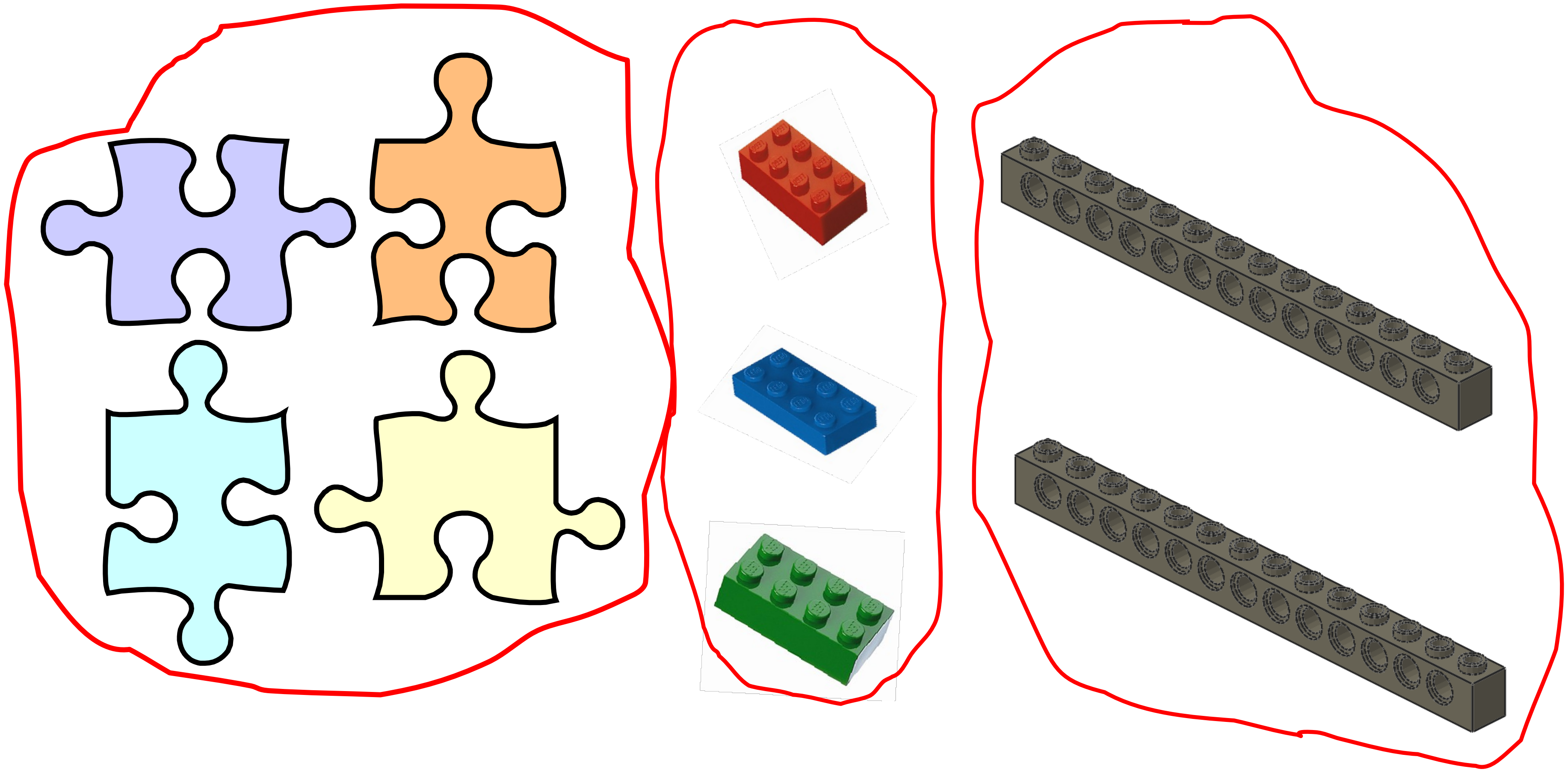
Software construction vs. creation: application is developed as an assembly of "integrated circuits"

Component based software construction – in practice



Component interactions

Components must obey to common conventions or standards !
Only in this way they will be able to recognise each others interfaces and connect and communicate to each other



CBSE essentials

- Independent components specified by their interfaces.
  - Separation between interface and implementation
  - Implementation of a component can be changed without changing the system
- Component standards to facilitate component integration.
  - Component models embody these standards
  - Minimum standard operations: how are interfaces specified, how communicate components
  - If components comply to standards, then their operation may be independent of their programming language
- Middleware that provides support for component inter-operability.
  - Provides support for component integration
  - Handles component communication, may provide support for resource allocation, transaction management, security, concurrency
- A development process that is geared to reuse.

CBSE and design principles

- Apart from the benefits of ***reuse***, CBSE is based on sound software engineering design principles that support the construction of ***understandable*** and ***maintainable*** software:
  - Components are independent so they do not interfere with each other;
  - Component implementations are hidden so they can be changed without affecting others;
  - Communication is through well-defined interfaces so if these are maintained one component can be replaced by another that provides enhanced functionality;
  - Component platforms (infrastructures) are shared and reduce development costs.

Component definitions - Szyperski

Szyperski:

"A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third-parties."

Component characteristics 1

Standardised	Component standardisation means that a component that is used in a CBSE process has to conform to some standardised component model. This model may define component interfaces, component meta-data, documentation, composition and deployment.
Independent	A component should be independent – it should be possible to compose and deploy it without having to use other specific components. In situations where the component needs externally provided services, these should be explicitly set out in a ‘requires’ interface specification.
Composable	For a component to be composable, all external interactions must take place through publicly defined interfaces. In addition, it must provide external access to information about itself such as its methods and attributes.

Fig. 19.1 from [Sommerville]

Component characteristics (cont)

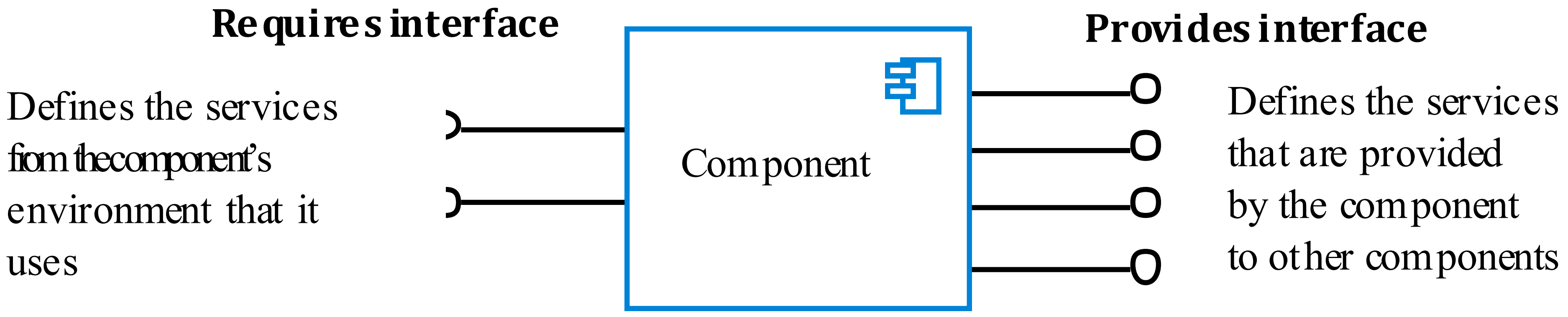
Deployable	To be deployable, a component has to be self-contained and must be able to operate as a stand-alone entity on some component platform that implements the component model. This usually means that the component is a binary component that does not have to be compiled before it is deployed.
Documented	Components have to be fully documented so that potential users of the component can decide whether or not they meet their needs. The syntax and, ideally, the semantics of all component interfaces have to be specified.

Fig. 19.1 from [Sommerville]

Component interfaces

- An interface of a component can be defined as a specification of its access point, offering no implementation for any of its operations.
- This seperation makes it possible to:
  - Replace the implementation part without changing the interface;
  - Add new interfaces (and implementations) without changing the existing implementation
- A component has 2 kinds of interfaces:
  - Provides interface
    - Defines the services that are provided by the component to the environment / to other components.
    - Essentially it is the component API
    - Mostly methods that can be called by a client of the component
  - Requires interface
    - Defines the services that specifies what services must be made available by the environment for the component to execute as specified.
    - If these are not available the component will not work. This does not compromise the independence or deployability of the component because it is not required that a specific component should be used to provide these services

Component interfaces



Requires interface
Defines the services from the component's environment that it uses
Component
Provides interface
Defines the services that are provided by the component to other components

Fig. 19.2 from [Sommerville]

Example: A data collector component

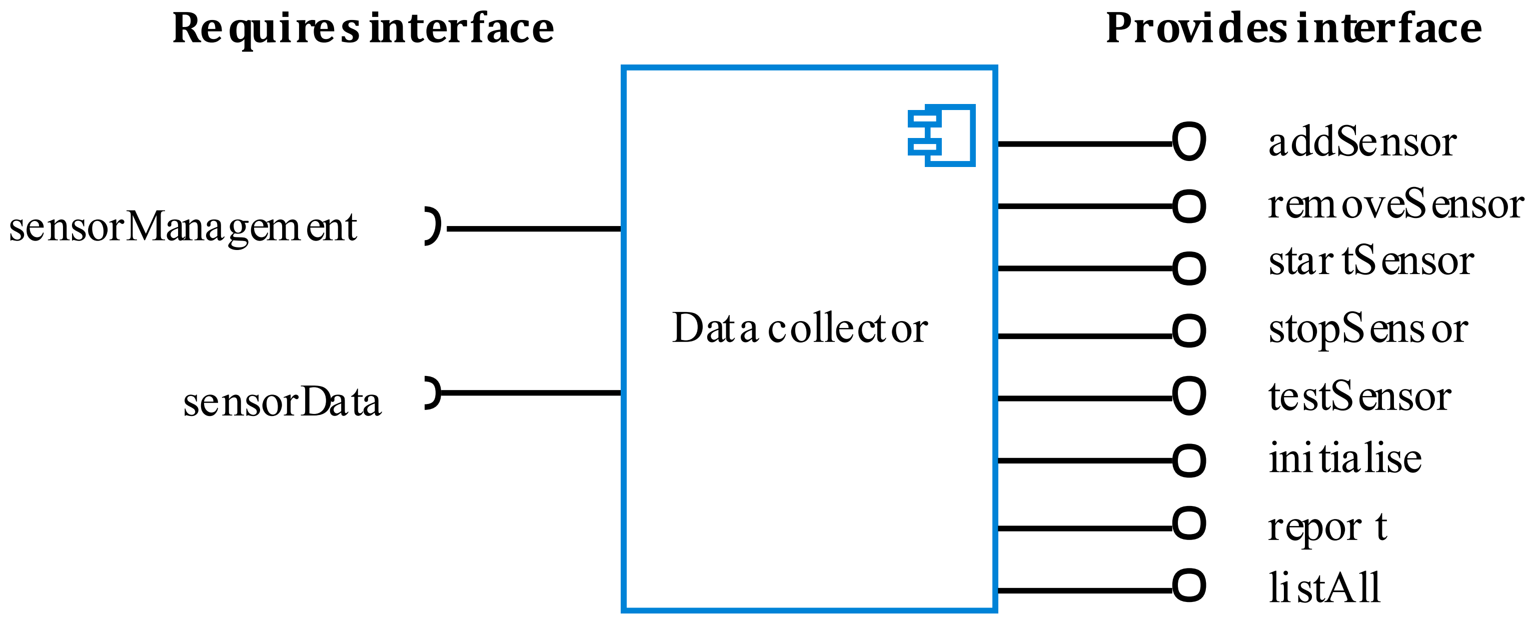
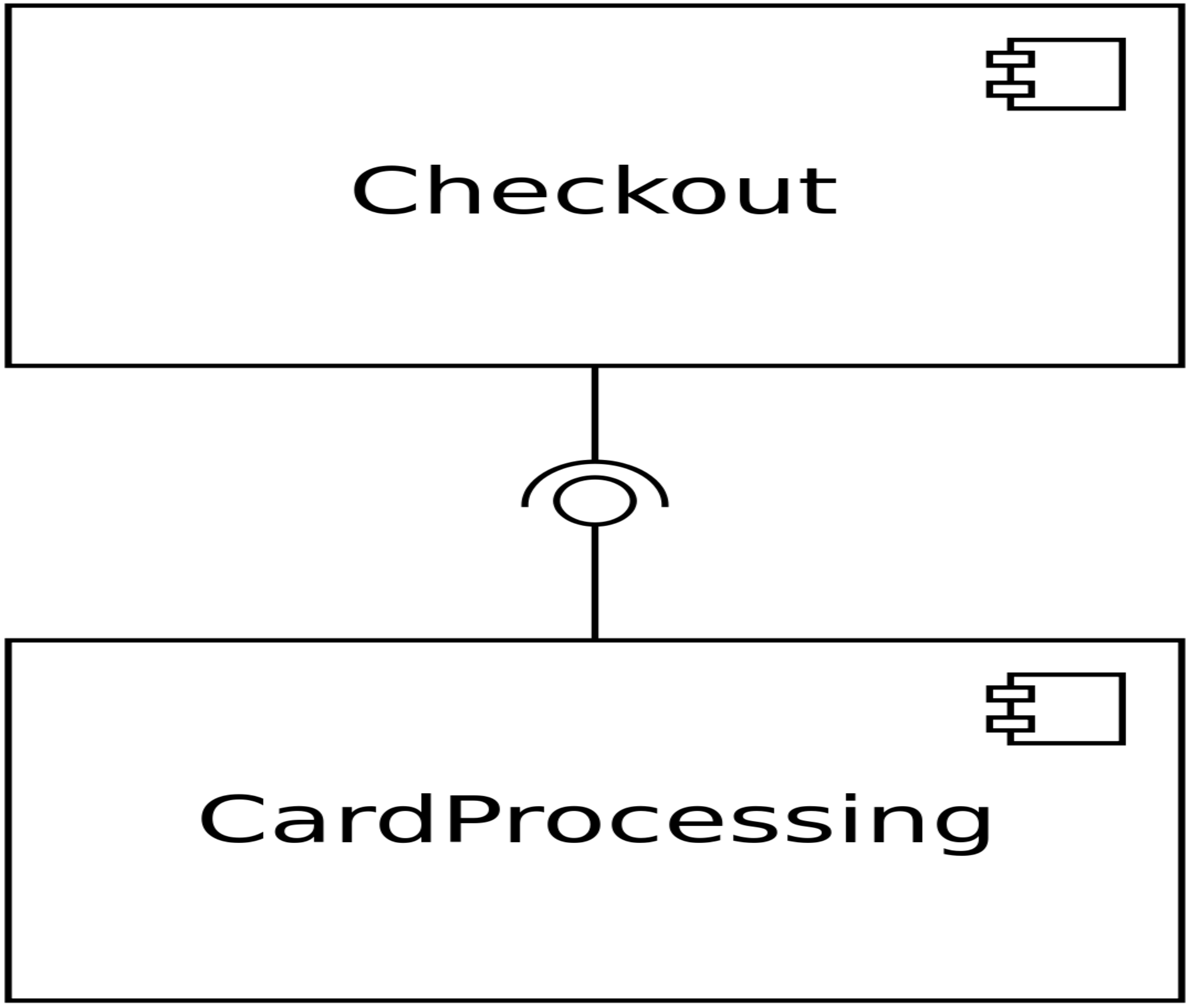
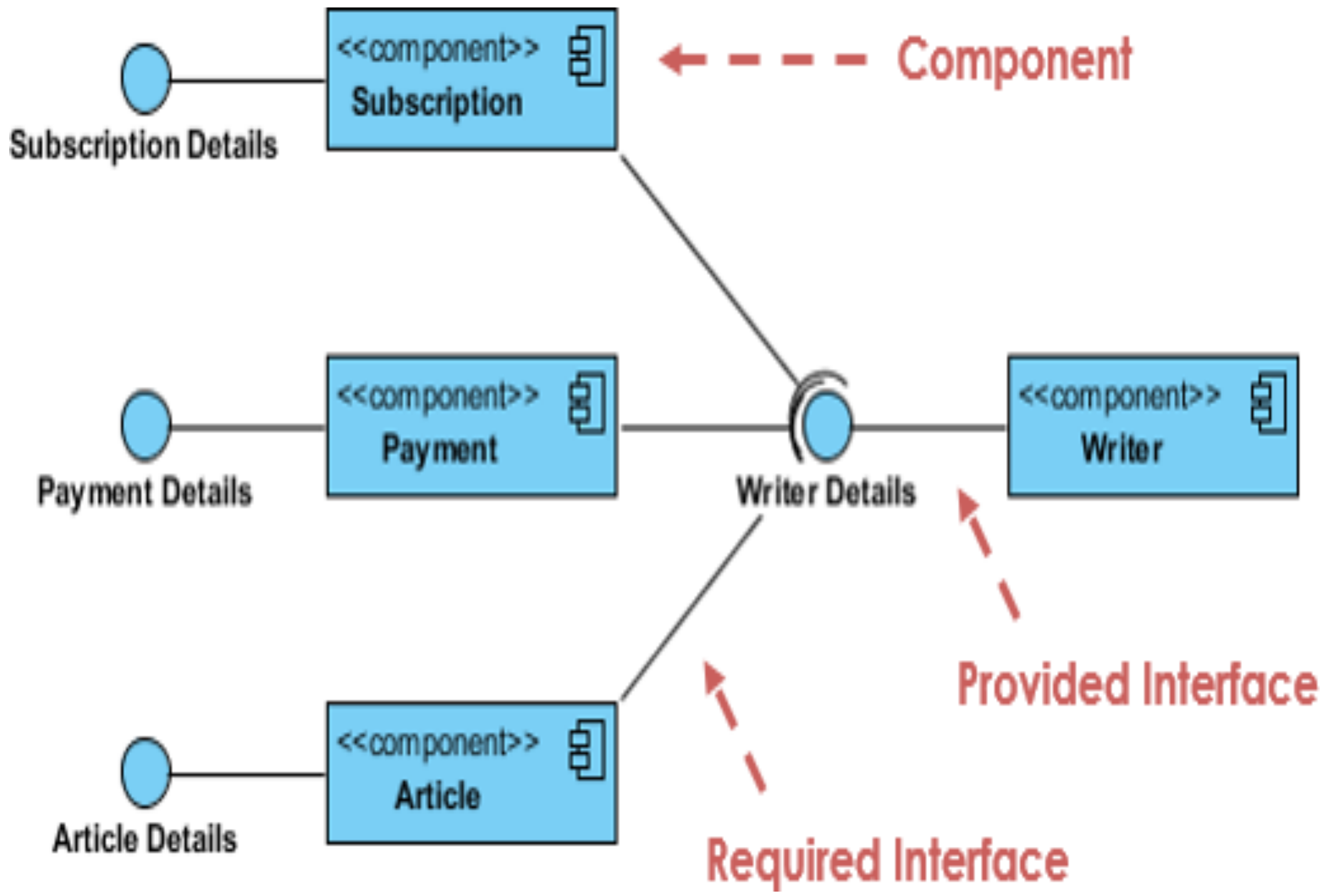
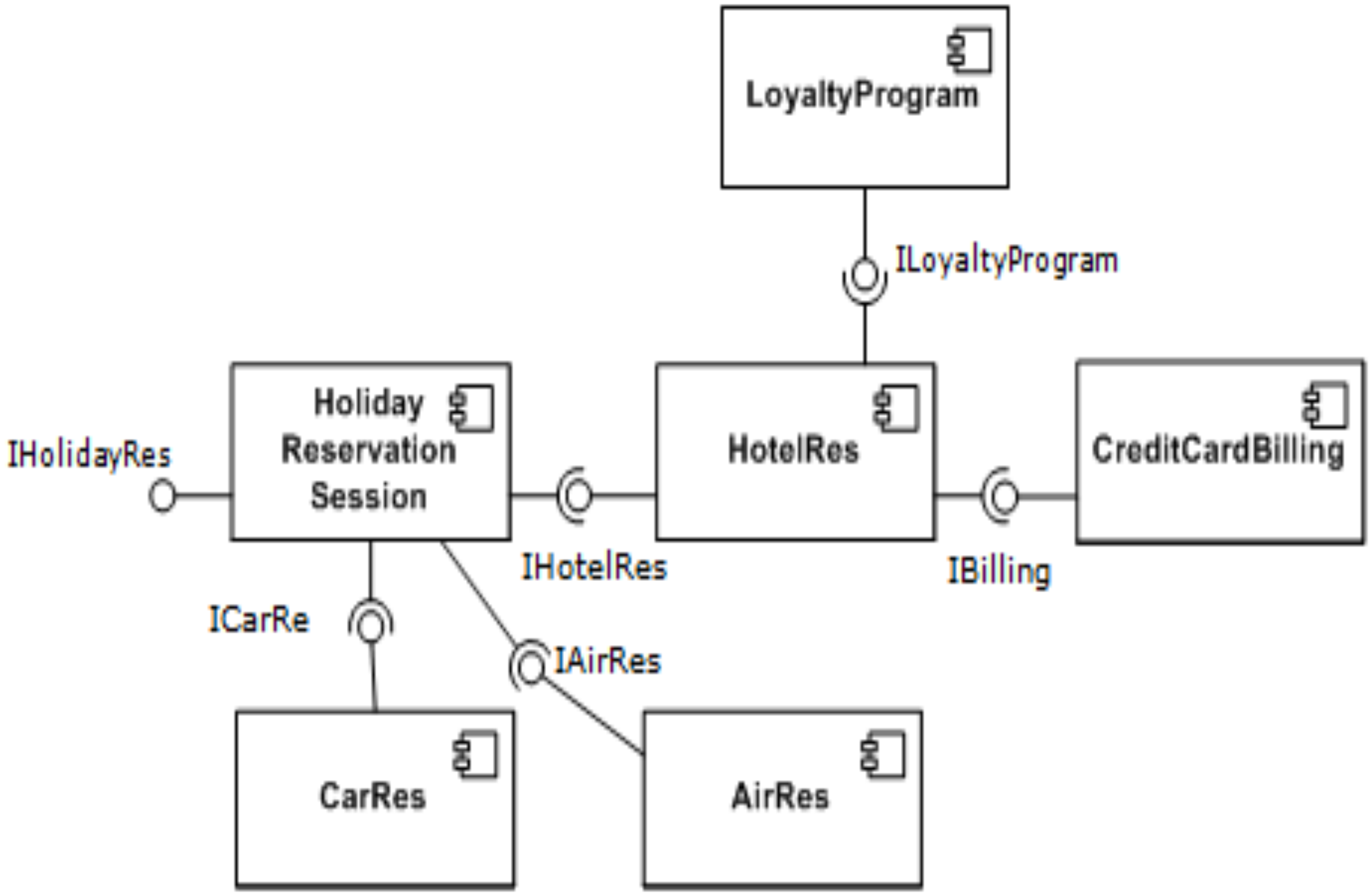


Fig. 19.3 from [Sommerville]


Checkout
CardProcessing



<<component>>
Subscription
Subscription Details
Component
<<component>>
Payment
Payment Details
Writer Details
<<component>>
Writer
<<component>>
Article
Article Details
Provided Interface
Required Interface


LoyaltyProgram
ILoyaltyProgram
IHolidayRes
Holiday Reservation Session
HotelRes
CreditCardBilling
IHotelRes
IBilling
ICarRe
IAirRes
CarRes
AirRes

Difference between Component and Object

- Components are deployable entities.
- Components do not define types.
- Component implementations are opaque.
- Components are language-independent.
- Components are standardised.

Describing interfaces

- Interfaces defined in standard component technologies using techniques such as Interface Definition Language (IDL) are:
  - Sufficient in describing functional properties.
  - Insuffiecient in describing extra-functional properties such as quality attributes like accuracy, availability, latency, security, etc.
- A more accurate specification of a component's behavior can be achieved through *contracts.*

Component models

- A component model is a definition of standards for component implementation, documentation and deployment.
- These standards are for:
  - component developers to ensure that components can interoperate
  - Providers of component executioninfrastructures who provide middleware to support component operation
- Examples of component models
  - EJB model (Enterprise Java Beans)
  - COM+ model (.NET model)
  - Corba Component Model
- The component model specifies how interfaces should be defined and the elements that should be included in an interface definition.

Component Based Development – Summary

- CBSE is about:
  - Building a system by composing “entities”
  - Reusing “entities”
  - Maintaining a system by adding/removing/replacing “entities”
- What are the “entities” ?
  - Functions, modules, objects, components, services, ..
- Reusable “entities” are encapsulated abstractions : provided/required interfaces